21世纪高等学校计算机应用技术规划教材



JSP动态网站开发基础与上机指导

范芸 范慧霞 主编
陈强 周雪敏 副主编

清华大学出版社

21世纪高等学校计算机应用技术规划教材

JSP动态网站开发基础与上机指导

范 芸 缪 亮 主编
陈 强 周雪敏 景 波 副主编

清华大学出版社
北 京

内容简介

本书系统地介绍了JSP技术的概念、方法和实现过程，包括JSP运行环境、JSP支持的体系结构、JSP的组成元素及内置对象、JSP对数据库及文件的操作、JSP对JavaBean的调用、JSP对Servlet的调用、JSP的框架应用等。通过本书的学习，读者可以系统地掌握JSP技术和相关概念、方法以及编程思路和技巧。

本书重点突出JSP编程思路和编程方法，以实例带动教学，注重对读者动手实践能力的培养。每章都在基础知识中间穿插"上机指导"教学单元，既可以让教师合理安排教学实践内容，又可以让学习者举一反三，快速掌握本章知识。

本书结构清晰、语言流畅、实例丰富，可作为各类院校计算机专业及相关专业的教材，也可作为培训机构相关专业的培训教材。

图书在版编目（CIP）数据

JSP动态网站开发基础与上机指导/范芸，范慧霞主编．—北京：清华大学出版社，2010.6
（21世纪高等学校计算机应用技术规划教材）
ISBN 978-7-302-22180-7

Ⅰ．①J… Ⅱ．①范… ②范… Ⅲ．①JAVA语言—主页制作—程序设计—高等学校—教材 Ⅳ．①TP393.092

中国版本图书馆CIP数据核字(2010)第033152号

责任编辑：魏江江 王冰飞
责任校对：白 蕾
责任印制：

出版发行：清华大学出版社 **地 址：**北京清华大学学研大厦A座
http://www.tup.com.cn **邮 编：**100084
社 总 机：010-62770175 **邮 购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经 销：全国新华书店
开 本：185×260 **印 张：**20.5 **字 数：**492千字
版 次：2010年6月第1版 **印 次：**2010年6月第1次印刷
印 数：1～ 000
定 价： .00元

产品编号：

编审委员会成员

（按地区排序）

浙江大学	吴朝晖	教授
	李善平	教授
扬州大学	李　云	教授
南京大学	骆　斌	教授
	黄　强	副教授
南京航空航天大学	黄志球	教授
	秦小麟	教授
南京理工大学	张功萱	教授
南京邮电学院	朱秀昌	教授
苏州大学	王宜怀	教授
	陈建明	副教授
江苏大学	鲍可进	教授
武汉大学	何炎祥	教授
华中科技大学	刘乐善	教授
中南财经政法大学	刘腾红	教授
华中师范大学	叶俊民	教授
	郑世珏	教授
	陈　利	教授
江汉大学	颜　彬	教授
国防科技大学	赵克佳	教授
中南大学	刘卫国	教授
湖南大学	林亚平	教授
	邹北骥	教授
西安交通大学	沈钧毅	教授
	齐　勇	教授
长安大学	巨永峰	教授
哈尔滨工业大学	郭茂祖	教授
吉林大学	徐一平	教授
	毕　强	教授
山东大学	孟祥旭	教授
	郝兴伟	教授
中山大学	潘小轰	教授
厦门大学	冯少荣	教授
仰恩大学	张思民	教授
云南大学	刘惟一	教授
电子科技大学	刘乃琦	教授
	罗　蕾	教授
成都理工大学	蔡　淮	教授
	于　春	讲师
西南交通大学	曾华燊	教授



随着我国改革开放的进一步深化，高等教育也得到了快速发展，各地高校紧密结合地方经济建设发展需要，科学运用市场调节机制，加大了使用信息科学等现代科学技术提升、改造传统学科专业的投入力度，通过教育改革合理调整和配置了教育资源，优化了传统学科专业，积极为地方经济建设输送人才，为我国经济社会的快速、健康和可持续发展以及高等教育自身的改革发展做出了巨大贡献。但是，高等教育质量还需要进一步提高以适应经济社会发展的需要，不少高校的专业设置和结构不尽合理，教师队伍整体素质亟待提高，人才培养模式、教学内容和方法需要进一步转变，学生的实践能力和创新精神亟待加强。

教育部一直十分重视高等教育质量工作。2007 年 1 月，教育部下发了《关于实施高等学校本科教学质量与教学改革工程的意见》，计划实施"高等学校本科教学质量与教学改革工程(简称'质量工程')"，通过专业结构调整、课程教材建设、实践教学改革、教学团队建设等多项内容，进一步深化高等学校教学改革，提高人才培养的能力和水平，更好地满足经济社会发展对高素质人才的需要。在贯彻和落实教育部"质量工程"的过程中，各地高校发挥师资力量强、办学经验丰富、教学资源充裕等优势，对其特色专业及特色课程(群)加以规划、整理和总结，更新教学内容、改革课程体系，建设了一大批内容新、体系新、方法新、手段新的特色课程。在此基础上，经教育部相关教学指导委员会专家的指导和建议，清华大学出版社在多个领域精选各高校的特色课程，分别规划出版系列教材，以配合"质量工程"的实施，满足各高校教学质量和教学改革的需要。

本系列教材立足于计算机公共课程领域，以公共基础课为主、专业基础课为辅，横向满足高校多层次教学的需要。在规划过程中体现了如下一些基本原则和特点。

(1) 面向多层次、多学科专业，强调计算机在各专业中的应用。教材内容坚持基本理论适度，反映各层次对基本理论和原理的需求，同时加强实践和应用环节。

(2) 反映教学需要，促进教学发展。教材要适应多样化的教学需要，正确把握教学内容和课程体系的改革方向，在选择教材内容和编写体系时注意体现素质教育、创新能力与实践能力的培养，为学生的知识、能力、素质协调发展创造条件。

(3) 实施精品战略，突出重点，保证质量。规划教材把重点放在公共基础课和专业基础课的教材建设上；特别注意选择并安排一部分原来基础比较好的优秀教材或讲义修订再版，逐步形成精品教材；提倡并鼓励编写体现教学质量和教学改革成果的教材。

(4) 主张一纲多本，合理配套。基础课和专业基础课教材配套，同一门课程可以有针对不同层次、面向不同专业的多本具有各自内容特点的教材。处理好教材统一性与多样化，基本教材与辅助教材、教学参考书，文字教材与软件教材的关系，实现教材系列资源配套。

（5）依靠专家，择优选用。在制定教材规划时依靠各课程专家在调查研究本课程教材建设现状的基础上提出规划选题。在落实主编人选时，要引入竞争机制，通过申报、评审确定主题。书稿完成后要认真实行审稿程序，确保出书质量。

繁荣教材出版事业，提高教材质量的关键是教师。建立一支高水平教材编写梯队才能保证教材的编写质量和建设力度，希望有志于教材建设的教师能够加入到我们的编写队伍中来。

21世纪高等学校计算机应用技术规划教材
联系人：魏江江 weijj@tup.tsinghua.edu.cn



近年来，JSP 技术得到了越来越广泛的应用，几乎所有基于 Java 的 Web 应用都使用了 JSP。JSP 集成了 Java 面向对象的程序语言能力与跨平台的优势，并且与超文本标记语言紧密结合，与传统的 CGI 程序相比，JSP 程序不仅编写、执行更容易，而且大幅提高了系统的执行性能。

本书面向初、中级用户，结合 JSP 和 Servlet 的最新规范，从基本的语法入手，以编程思路为主线，以应用为目标，运用实例系统由浅入深地阐述了如何运用 JSP 开发 Web 应用程序。

主要内容

本书以精简的内容介绍了 JSP 的语法、Servlet 技术、JDBC 技术、表达式语言、Struts 技术等。全书共分 10 章，各章具体内容介绍如下：

第 1 章学习 JSP 基础知识、JSP 的优缺点、安全性和应用前景。

第 2 章学习 JSP 的运行环境及安装配置方法。

第 3 章学习 JSP 的语法基础。

第 4 章学习 JSP 内置对象的概念、作用和使用方法。

第 5 章学习 JSP 页面对数据库的操作方法。

第 6 章学习 JSP 页面对文件的操作方法。

第 7 章学习 JavaBean 技术及 JSP 页面对 JavaBean 的使用方法。

第 8 章学习 Servlet 编程方法和 JSP 页面调用 Servlet 的方法。

第 9 章学习 Struts 的基本结构及工作流程，以及如何应用 Struts 架构。

第 10 章通过 3 个综合编程实例，学习运用 JSP 技术解决实际问题的方法和技巧。

本书特点

1. 紧扣教学规律，合理设计图书结构

本书作者多是长期从事 JSP 教学工作的一线教师，具有丰富的教学经验，紧扣教师的教学规律和学生的学习规律，全力打造难易适中、结构合理、实用性强的教材。

本书采取“知识要点－基础知识讲解－典型实例讲解－上机指导－习题”的内容结构。在每章的开始给出本章的主要内容简介，读者可以了解本章所要学习的知识点。在具体的教学内容中既注重基本知识点的系统讲解，又注重学习目标的实用性。每章都设计了“本章习题”，既可以让教师合理安排教学内容，又可以让学习者加强实践，快速掌握本章知识。

2. 注重教学实验，加强上机指导内容的设计

JSP 技术是一门实践性很强的课程，学习者只有亲自动手上机练习，才能更好地掌握教

材内容。本书将上机练习的内容设计成“上机指导”教学单元,教师可以根据课程要求灵活授课和安排上机实践。读者可以根据上机指导中介绍的方法、步骤进行上机实践,然后根据自己的情况对实例进行修改和扩展,以加深对其中所包含的概念、原理和方法的理解。

3. 理论联系实际,实例贯穿知识点

本书自始至终都以实例引导知识点的学习,通过实例来理解概念,通过应用来熟悉技术,理论与实际相结合,使读者易学易用,学以致用。书中实例都是以最新标准为基础,介绍JSP的最新发展,且在实际开发工作中经常碰到的问题在案例中都有体现,更贴近实用。

4. 专设图书服务网站,打造知名图书品牌

为了帮助读者建构真正意义上的学习环境,以图书为基础,为读者专设一个图书服务网站。网站提供相关图书资讯,以及相关资料下载和读者俱乐部。在这里读者可以得到更多、更新的共享资源,还可以交到志同道合的朋友,相互交流、共同进步。

网站地址:http://www.cai8.net。

本书构思科学合理,理论与应用配合紧密,语言通俗易懂,既可作为各类院校计算机专业及相关专业的教材,也可以作为培训机构相关专业的培训教材。

本书作者

本书作者曾从事多年的计算机应用系统设计和开发工作,积累了丰富的编程思想和编程方法,近几年从事高校教学工作,既有丰富的系统开发经验,又有丰富的教学经验,是主讲Java技术和JSP技术的一线教师。

本书主编为范芸(负责编写第2章~第6章,提纲设计、稿件初审、前言编写等)、范慧霞(河北农业大学中兽医学院,负责编写第10章),副主编为陈强(负责编写第7章~第9章)、周雪敏(负责编写第1章及电子课件制作)。

在本书的编写过程中,关南宝、叶君、胡彦玲、谭晓芳、雷学锋、杨梅、李捷、李泽如、许美玲、张爱文、赵崇慧、郭刚、朱桂红、李敏等参与了本书实例制作和部分内容编写的工作,在此表示感谢。另外,感谢南昌理工学院对本书的创作和出版给予的支持和帮助。

由于编写时间有限,加之作者水平有限,疏漏和不足之处在所难免,恳请广大读者批评指正。

作　者

2010年3月

目 录

第 1 章 JSP概述

JSP 是 Java Server Pages 的缩写，是由 Sun 公司倡导，多家公司参与一起建立的一种应用范围非常广泛的动态网页技术标准，是目前 Web 开发技术中应用最广泛的动态网页技术之一。本章将对 JSP 技术及其优缺点、应用前景作简要介绍。

本章主要内容：

- JSP 技术简介；
- JSP 的优点与缺点；
- JSP 技术的发展前景；
- JSP 的安全性。

1.1 动态网页技术

随着 Internet 进入人们的生活，Web 已经不可能再将其功能局限于静态的信息发布平台。今天的 Web 已经可以提供个性化搜索的功能，如可以收发电子邮件，可以进行网上销售，可以从事电子商务等，而这些功能的实现都必须使用更新的网络编程技术来制作动态网页。

所谓动态，指的并不是包含 Flash 或 Gif 文件那种可以动的网页，而是可以根据访问者的不同需要，对访问者输入的信息提供不同的响应的网页。这就意味着，不同的人、不同的时间、不同的输入访问同一网址时会得到不同的页面。虽然，客户端用户所接收到的页面与传统页面并没有任何区别，但实际上这些页面内容已经经过了服务器端的处理，下面介绍几种常见的动态网页技术。

1.1.1 ASP 技术

ASP(Active Server Pages)是微软开发的一种类似 HTML(超文本标识语言)、Script(脚本)与 CGI(公用网关接口)的结合体，它没有提供自己专门的编程语言，而是允许用户使用许多已有的脚本语言来编写 ASP 的应用程序。ASP 的程序编制比 HTML 更方便且更具灵活性，它在 Web 服务器端运行，运行后再将运行结果以 HTML 格式传送至客户端的浏览器。因此，ASP 与一般的脚本语言相比，要安全得多。

ASP 的最大好处是可以包含 HTML 标签，也可以直接存取数据库及使用无限扩充的 ActiveX 控件，因此在程序编制上要比 HTML 方便而且更富有灵活性。通过使用 ASP 的

组件和对象技术，用户可以直接使用 ActiveX 控件，调用对象方法和属性，以简单的方式实现强大的交互功能。

但 ASP 技术也并非完美无缺，由于它基本上是局限于微软的操作系统平台之上，主要工作环境是微软的 IIS 应用程序结构，又因 ActiveX 对象具有平台特性，所以 ASP 技术不能很容易地实现在跨平台 Web 服务器上工作。

1.1.2 PHP 技术

PHP 是一种跨平台的服务器端的嵌入式脚本语言。它大量地借用 C、Java 和 Perl 语言的语法，并耦合 PHP 自己的特性，使 Web 开发者能够快速地写出动态产生的页面。它支持目前绝大多数数据库。还有一点，PHP 是完全免费的，不用花钱购买，可以从 PHP 官方站点(http://www.php.net)自由下载。而且还可以不受限制地获得源码，甚至可以加入用户自己需要的特色。

PHP 技术与 HTML 语言具有非常好的兼容性，使用者可以直接在脚本代码中加入 HTML 标签，或者在 HTML 标签中加入脚本代码，从而更好地实现页面控制。PHP 提供了标准的数据库接口，数据库连接方便、兼容性强、扩展性强，可以进行面向对象编程。

1.1.3 Servlet 技术

Servlet 技术是 Sun 公司提供的一种实现动态网页的解决方案，是基于 Java 编程语言的 Web 服务器端编程技术，主要用于在 Web 服务器端获得客户端的访问请求信息和动态生成对客户端的响应消息。Servlet 技术也是 JSP 技术(另外一种动态网页开发技术)的基础。一个 Servlet 程序就是一个实现了特殊接口的 Java 类，用于被支持 Servlet 的 Web 服务器调用和运行，即只能运行于具有 Servlet 引擎的 Web 服务器端。一个 Servlet 程序负责处理它所对应的一个或一组 URL 地址的访问请求，接收访问请求信息和产生响应内容。

1.2 JSP 简介

JSP(Java Server Pages)是由 Sun 公司倡导，多家公司参与，于 1999 年推出的一种动态网页技术标准，是基于 Java Servlet 以及整个 Java 体系的 Web 开发技术。在技术上，JSP 有点类似 ASP，它是在传统的网页 HTML 文件中插入 Java 程序段和 JSP 标记，从而形成 JSP 文件。

1.2.1 JSP 的工作原理

JSP 技术使用 Java 编程语言编写类 XML 的 tags 和 Scriptlets，它与 Java Servlet 一样，是在服务器端执行的，通常返回该客户端的就是一个 HTML 文本，因此客户端只要有浏览器就能浏览。JSP 的工作原理如图 1-1 所示。

(1) 当用户访问一个 JSP 页面时，用户从客户端浏览器向服务器发出一个 JSP 页面请求，这些请求里面有很多信息，包括请求的文件、用户输入的内容及一些本地计算机的信息。

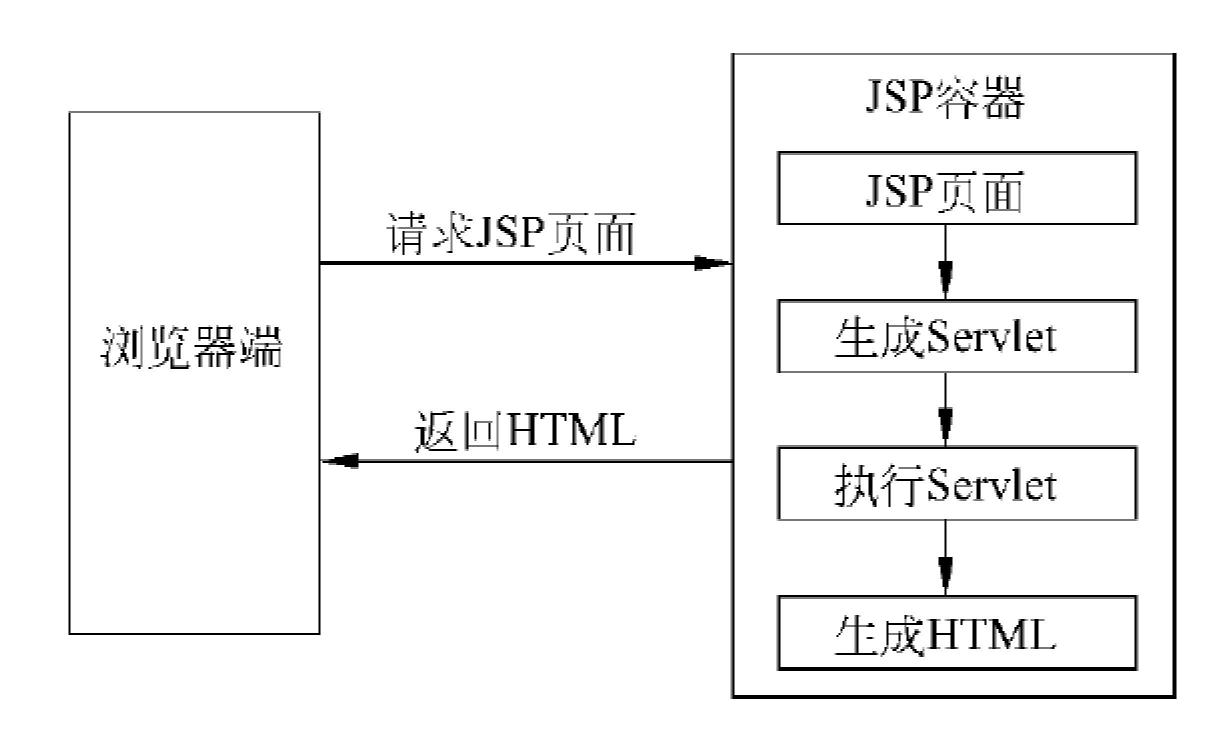


图 1-1 JSP 的运行原理

（2）JSP 文件由 JSP 引擎将 JSP 源代码转换成 Servlet 代码文件。

（3）由 JSP 引擎调用服务器端的 Java 编译器对 Servlet 代码进行编译，由此生成字节码文件（.class）。

（4）由 Java 虚拟机执行此字节码文件，并将执行结果以 HTML 格式发送到请求浏览器。

（5）由浏览器对这些 HTML 代码进行解释，并将结果显示在浏览器窗口中。

【说明】 只有在第一次请求 JSP 页面时，才需要进行上述转换和编译，此时运行速度比较慢。如果以后再次请求该页面，则会直接运行第一次请求时生成并保存在服务器端的字节码文件，运行速度将显著加快。如果对 JSP 源文件进行了修改，则当收到对此 JSP 页面的请求时，JSP 引擎将重新对源文件进行转换和编译，并利用新生成的字节码文件覆盖原来的文件。

1.2.2 JSP 的优势与劣势

JSP 技术主要有以下的优势。

1. 一次编写，处处运行

众所周知，由于微软的垄断性，它的产品可移植性十分差，ASP 也不例外，在 Windows 平台下编写的 ASP 代码，很难在别的平台上运行，相反，JSP 使用的是 Java 语言，它继承了 Java 语言的特点——“一次编写，处处运行”，这种概念正越来越深远地影响着因特网行业的交互式 Web 页面的设计理念。JSP 页面可以非常容易地跨平台、跨 Web 服务器软件来设计和开发源代码。

2. JSP 组件跨平台

JSP 组件（企业 JavaBeans、JavaBeans 或定制的 JSP 标签）都是跨平台可重用的。企业 JavaBeans 组件可以访问传统的数据库，并能以分布式系统模式工作于 Solaris、Linux、UNIX 和 Windows 平台。

3. 强大的可伸缩性

从只有一个小的 Java 文件就可以运行 Servlet/JSP，到由多台服务器进行集群和负载

均衡,再到多台 Application 进行事务处理、消息处理,从一台服务器到无数台服务器,Java 显示了巨大的生命力。

4. 支持多种网页格式

目前,JSP 技术支持的网页格式还没有一个明确的标准。一般来说,JSP 技术既可以支持 HTML/DHTML 的传统浏览器文件格式,又可以支持应用于无线通信设备如移动电话、PDA 等设备进行网页预览的 WML 文件格式,还可以支持其他一些 B2B 电子商务网站应用的 XML 格式。

JSP 技术的劣势表现在以下几个方面。

(1) 与 ASP 一样,Java 的一些优势正是它致命的问题所在。正是为了跨平台的功能,为了极度的伸缩能力,所以极大地增加了产品的复杂性。Java 系统开发了多种产品,如 JRE、JDK、J2EE、EJB、JSWDK 和 JavaBeans,只有有效地将它们组合在一起,才能产生强大的功能。

(2) Java 的运行速度是用 class 常驻内存来完成的,所以它在一些情况下所使用的内存比起用户数量来说确实是“最低性能价格比”了。从另一方面来说,它还需要硬盘空间来储存一系列的.java 文件和.class 文件,以及对应的版本文件。

(3) JSP 程序调试很困难,JSP 页面执行时,首先被转换为.java 文件(Servlet),然后将.java 文件编译为字节码文件。这样,出错信息实际上指向的是转换后的那个.java 文件(Servlet),而不是 JSP 本身。

1.2.3 JSP 的技术前景

JSP 是基于 Java 的技术,它具备了 Java 语言所有的优点,同时拥有强大的服务器端动态网页技术功能。目前在国内,PHP 与 ASP 应用最为广泛,而 JSP 由于是一种较新的技术,国内采用得较少。但在国外,JSP 已经是比较流行的一种技术,尤其是电子商务类的网站,多采用 JSP。在这三者中,JSP 应该是未来发展的趋势。世界上一些大的电子商务解决方案提供商都采用 JSP/Servlet 方案。比较出名的如 IBM 的 E-business,它的核心是采用 JSP/Servlet 的 WebSphere,它们都是通过 CGI 来提供支持的。

1.3 JSP 的安全性

JSP 编程语言自从推出之日起,由于它的快速、平台无关、可扩展、面向对象等特性得到了越来越广泛的应用,越来越多的厂家开发出了各种各样的支持平台如 IBM 公司的 WebSphere、BEA 公司的 WebLogic 等,也有越来越多的网站开始将自己的平台架构在 JSP 环境中。

但随着网络编程越来越方便,系统功能越来越强大,系统的安全性问题也越来越严重。为了给用户提供更多、更强大的服务,目前大多数网站都选择动态生成网页内容。这虽然大大促进了 WWW 的发展,但同时也给系统设计师和开发者提出了网络安全的问题。

1.3.1 JSP 安全性的实现方法

为了防止被黑客攻击从而保证系统的正常运行，JSP 采取了很多有效的措施来提高 Web 应用程序的安全性。JSP 中的安全性主要通过以下几种方式来实现。

1. Declarative Security

Declarative Security 指的是表达一个应用的安全结构，包括角色、存取控制和在一个应用的外部表单所要求的验证。在 Web Application 中发布描述器，是实施 Declarative Security 的一种主要工具。发布者把 Application 所要求的逻辑完整性映射为在特定运行环境下的安全策略。在运行时，Servlet Container 使用这些策略来强迫验证。

2. Programmatic Security

当 Declarative Security 不能够完全表达一个 Web Application 的安全模型时，就可以使用 Programmatic Security。Programmatic Security 包括 HttpServletRequest 接口的 3 种方法：GetRemoteUser、IsUserInRole 和 GetUserPrincipal。

GetRemoteUser 方法返回经过客户端验证的用户名；IsUserInRole 向 Container 的安全机制询问一个特定的用户是否在一个给定的安全角色中；GetUserPrincipal 方法返回一个 Java.Security.Principal 对象。这些应用程序接口方法根据远程用户的逻辑角色让 Servlet 去完成一些逻辑判断。

3. Roles

一个 Roles 就是由 Application Developer 和 Assembler 所定义的一个抽象的逻辑用户组。当一个 Web Application 被发布时，Developer 就把这些角色映射到在运行环境中的安全认证中，例如组或规则。一个 Servlet Container 能够为规则执行一些说明或编程安全，这些规则是与调用这些 Principal 安全属性所输入的要求相联系的。

4. Authentication

一个 Web 客户端能够使用 4 种机制来对 Web 服务器进行用户验证，这 4 种机制分别为 HTTP Basic Authentication、HTTP Digest Authentication、HTTPS Client Authentication 和 HTTP Based Authentication。

5. HTTP Basic Authentication

HTTP Basic Authentication 是一个定义在 HTTP/1.1 规范中的验证机制，这种机制是以用户名和密码为基础的。一个 Web Server 要求一个 Web Client 去验证一个用户。作为 Request 的一部分，Web Server 传递被称为 realm 的字符串，用户就是在它里面被验证的。

6. HTTP Digest Authentication

与 HTTP Digest Authentication 一样，HTTP Digest Authentication 根据用户名和密

码验证一个用户，然而密码的传输是通过一种加密的形式进行的，这就比 Basic Authentication 所使用的 64 位编码形式传递要安全得多。由于 Digest Authentication 当前不被广泛使用，因此 Servlet Containers 不要求支持它，但是鼓励去支持它。

7. HTTPS Client Authentication

使用 HTTPS(HTTP over SSL)的终端用户验证是一种严格非验证机制。这种机制要求用户去处理公共密钥证明(Public Key Certification，PKC)。当前，PKCs 在 e-commerce 应用中是有用的。不适应 J2EE 的 Servlet Containers 不要求支持 HTTPS 协议。

JSP 主要通过以上的措施最大限度地保证 Web 应用的安全，但是在实际的应用过程中，由于 JSP 自身的局限、网络传输和操作系统等原因，JSP 开发的 Web 应用系统仍然存在着安全性问题。

1.3.2 源代码暴露问题

源代码暴露类别主要指的是程序源代码会以明文的方式返回给访问者。从理论上讲，不管是 JSP 还是 ASP、PHP 等动态程序都是在服务器端执行的，程序执行后只返回给访问者标准的 HTML 代码，由于服务器内部机制的问题在运行时可能会引起源代码暴露的漏洞。引起源代码暴露的情况主要有以下几种。

1. 添加特殊后缀引起 JSP 源代码暴露

例如：Tomcat3.1 下，在浏览器的地址栏中输入 http://localhost:8080/index.jsp，可以正常解释执行，将 index.jsp 改为 index.JSP 或者 index.Jsp 等时，浏览器会提示下载这个文件，下载后就可得到源代码。

出现这种情况的原因在于 JSP 是区分大小写的，而 Tomcat 只会将小写的 jsp 后缀的文件当作是正常的 JSP 文件来执行，如果大写了就会引起 Tomcat 将 index.JSP 当作是一个可以下载的文件让客户下载。老版本的 WebLogic、WebSphere 等都存在这个问题，现在这些公司发布了新版本或者补丁解决了这问题。

2. 插入特殊字符串引起 JSP 源代码暴露

还有一种是插入特殊字符串引起的漏洞，BEA WebLogic Enterprise 5.1 文件路径开头为“/file/”的漏洞、IBM WebSphere 3.0.2 文件路径开头为“/servlet/file/”的漏洞等。

如在 IBM WebSphere 3.0.2 中，若一个请求文件的 URL 为“login.jsp”：http://site.running.websphere/login.jsp，当把 URL 改为 http://site.running.websphere/servlet/file/login.jsp 将看到这个文件的源代码。

出现该情况的原因是，IBM WebSphere 3.0.2 是调用不同的 Servlet 对不同的页面进行处理，如果一个请求的文件是未进行注册管理的，WebSphere 会使用一个默认的 Servlet 进行调用。如果文件路径以“/Servlet/file/”作为开头，这个默认的 Servlet 会被调用这个请求的文件不经过分析或编译就显示出来。而解决这种问题的方法就是在服务器软件的网站下载最新的补丁。

3. 路径权限引起的文件 JSP 源代码暴露

大部分的 JSP 应用程序在当前目录下都会有一个 WEB-INF 目录，这个目录通常存放的是 JavaBeans 编译后的 class 文件，如果不给这个目录设置正常的权限，所有的 class 就会曝光。

例如，若采用的是 Apache 1.3.12 加上第三方 JSP 软件形式的 Web 服务器，因为 Apache 1.3.12 默认的设置是可以读取目录的，如果程序的 URL 是 http://site.running.websphere/login.jsp，当在浏览器的地址栏中输入 http://site.running.websphere/WEB-INF/时，就可以看到这个目录下以及这个目录下的子目录中的 class 文件，还可以下载到本机上。

在 JSP 环境下，同样可以通过设置服务器的环境来解决这个问题，简单地说就是将一些比较重要的目录如 WEB-INF、classes 等设置访问权限，不允许读取只允许执行。以 Apache 为例，可以在 httpd.conf 文件中添加一个目录 WEB-INF 并设置 Deny from all 等属性。

4. 文件不存在引起的绝对路径暴露问题

很多人都知道，微软 IIS 5.0 中也有比较多的 *.idc 暴露绝对路径漏洞。同样这些问题现在也转到了 JSP 环境中，这个漏洞暴露了 Web 程序的绝对硬盘地址，和其他漏洞结合就具有比较大的危害了，因为这个漏洞目前还没有在国外安全网站上看到，如果黑客将其和其他漏洞结合进行攻击，就非常危险了。

例如，在特定的服务器软件下，访问一个不存在的 JSP 文件如 http://localhost:8080/fdasfas.jsp，就会返回 java.servlet.ServletEception：java.io.FileNotFoundEception：c：webappfadssad.jsp (???????????)这样的错误，这样就可以知道网站在 C:\web\app 目录下，也许一般人不太在意，但是对于一个黑客来说就很有帮助了。

出现这种情况的原因是负责 JSP 执行的相关 Servlet 中处理异常的时候没有过滤掉这种情况，而解决方法就是下载最新的补丁。如果当时的 Web 服务器软件没有这个补丁，可以找到服务器软件的 JSP 执行映射 Servlet 文件(当然是 class 后缀的)，将它用 JAD 软件进行反编译，在反编译后的源代码中找到处理 Eception 的方法，然后将方法中的处理部分全部注释掉，并将请求导向到一个自定义的出错页面中，这样问题就解决了。

1.3.3 其他问题

除了前面介绍的源代码暴露问题外，JSP 在远程程序执行和其他方面也存在着安全漏洞问题，而这些安全问题也都直接影响到这个系统的正常运行。

1. 远程程序执行问题

远程程序执行问题的特点是可以通过 URL 地址在浏览器中执行任意服务器上的命令和程序，从而引起安全问题。

如果 URL 请求的目标文件使用了前缀/Servlet/，则 JSP 解释执行功能被激活。这时在用户请求的目标文件路径中使用“../”，就有可能访问到 Web 服务器上根目录以外的文

件。如果目标主机上利用该漏洞请求用户输入产生的一个文件,就会严重威胁目标主机系统的安全,而解决这类问题的方法就是安装最新的补丁。

2. 其他问题

除 JSP 本身存在安全漏洞问题以外,其他和 JSP 应用相关的工具也会给 JSP 应用带来安全问题,如数据库(SQL Server、Oracle、DB2 等)的漏洞,操作系统(Windows NT/2000、Linux 等)的漏洞。这些漏洞可以说都是致命的,如利用 Linux 的某些漏洞可以轻易地获得管理员权限来远程控制服务器,获得系统的完全控制权限。

JSP 还存在着很多安全上的问题。一般来说,服务器软件的开发商在内部测试中不可能将系统中的所有 bug 都找出来,即使发布软件后,被发现的漏洞也只会是其中的很小一部分,所以程序开发人员必须时刻提高警惕,注意系统安全。

本章小结

JSP 是一种目前较为流行的基于 Java Servlet 的 Web 开发技术。本章首先介绍动态网页技术,然后讲解动态网页技术的 3 种技术的比较,随后着重介绍了 JSP 技术的原理和 JSP 的安全性问题,主要为后面的学习打下基础。

习题 1

1. 简述 ASP、PHP 和 Servlet 技术的比较。
2. 简述 JSP 的特点及工作原理。
3. 简述 JSP 的安全性实现方法。
4. 简述 JSP 的安全漏洞问题。

第2章 JSP运行环境的搭建

JSP是一种执行于服务器端的动态网页开发技术，为了编写和调试JSP程序，需要建立一个JSP的运行和开发环境。根据JSP的运行原理，JSP的运行离不开Web服务器的支持，而对JavaBean及Servlet进行编译和运行必须有Java JDK的支持，所以建立JSP运行环境首先要安装JDK以及支持JSP的Web服务器。本章主要介绍如何安装和配置JSP页面运行环境。

本章主要内容：

- JSP的开发工具；
- 安装和配置Java开发包(JDK)；
- 安装和配置Tomcat服务器。

2.1 JSP开发工具简介

随着JSP技术的不断发展和广泛应用，很多公司都推出了JSP的开发工具，读者可以参照其各自的特点，结合自身开发环境进行开发工具的选择。本节着重介绍JSP轻量级开发的运行环境，以及开发的经典模式MVC。

2.1.1 JSP运行的最佳环境——Tomcat+MySQL

Tomcat是一个免费的Web应用服务器，也就是常说的JSP运行容器。MySQL也是免费的数据库服务器，从一开始就定位在快速稳定的大型关系型数据库上，因此，其性能和稳定性相比于其他的开源数据库占有绝对优势。

之所以称此二者为最佳运行环境，主要原因在于，首先它们是免费的，并且有无数人齐心协力对其进行长久的优化，有很多的优秀论坛和热心用户组，还有很多的成功案例供参考咨询。其次，它们对系统的要求较低，可以在不同的操作系统下运行，而且它们在性能上的表现基本可以满足一般应用系统的要求。最后，配置和维护Tomcat及MySQL都十分简单，直接修改配置文件就可以配置运行环境的各种特性，复制文件系统即可完成系统的备份，非常适用于远程网络的环境。

2.1.2 高效开发JSP的最佳搭配工具——Eclipse+MyEclipse

在开发工具方面，IBM、Borland、Sun等软件巨头在其应用服务器的基础上，都推出了开发Java Web应用程序的开发工具，如WSAD、Borland Jbuilder、Sun ONE Studio、BEA

Workshop 等。这些 IDE 大多都兼顾了各种类别的 Java 应用程序的开发,需要较大的空间进行安装,启动过程也相对较慢。而且,由于它们都是各厂商的私有产品,外界无法对其进行定制或者改造,因此无法跟随 J2EE 不断前进的脚步。

Eclipse 是由一群无私的开发者开发的,作为目前 IDE 的佼佼者,Eclipse 因其开放性受到了越来越多用户及厂商的欢迎。

首先,Eclipse 是免费的,遵循 Common Public License 协议。用户可以免费获取 Eclipse 软件本身及其源代码。其次,Eclipse 是免安装的,只要本机装有 JDK,就可以将其他机器上的 Eclipse 目录复制到本机,然后经过简单的配置就可以使用。

另外,Eclipse 是开放的,由于其设计的精妙,任何人或厂商都可以编写自己的插件,并将其商业化。因此,Eclipse 理论上可以做任何事情,而不仅仅是一个 Java 的 IDE。用户甚至可以在 Eclipse 上编写 C++代码,建模、查询不同数据库的数据。

但在 Eclipse 标准软件包中只提供了 Java 应用程序的开发和调试,而没有提供 JSP 开发环境,不过有众多的插件支持在 Eclipse 上进行 JSP 开发,而其中最强大的非 MyEclipse 莫属了。

MyEclipse 具备了众多令人欣喜的特色,贴心的 Wizard、图形化的配置管理、JSP 错误跟踪、代码跳转等,都可以令 J2EE 开发飞速运转起来。MyEclipse 创建的工程还支持若干开源框的注入,如 Struts、Spring 和 Hibernate 等,紧跟前沿开发潮流。

本书推荐读者采用 Eclipse+MyEclipse 进行 Web 开发。

2.1.3 开发 JSP 的经典模式 MVC

Java 之所以受到厂商的追捧,是因为其跨平台的特性。而 Java 应用程序开发之所以受到广大开发人员的青睐,最重要的原因是其开放性和优雅的设计。而其中最令人赞叹的就是 MVC 理论在 Java 领域的生根发芽。近年来,随着 J2EE 技术的成熟,MVC 逐渐成为了备受推崇的设计模式。

MVC 是 Model View Controller 的缩写,概括了应用程序开发的 3 个重要角色(模型对象、表现形式和流程控制)之间的关系。应用程序的输入、处理和输出流程按照 3 种角色划分为三层,即模型层、视图层和控制层。图 2-1 直观地表示了 MVC 架构的内部关系。

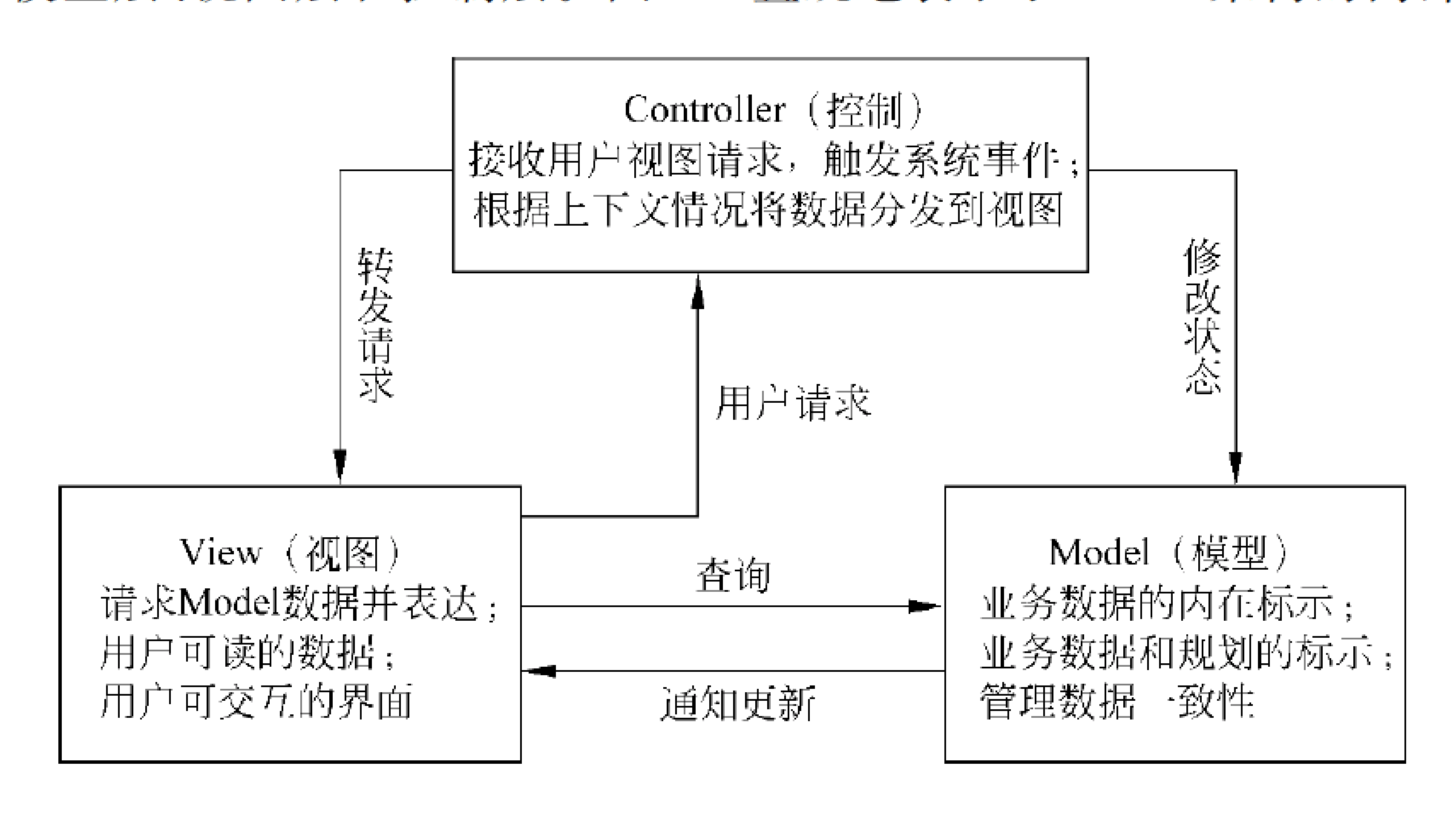


图 2-1 MVC 架构的内部关系

实际上，这3个角色在许多应用程序中都存在，模型对象负责数据的表征，视图负责应用程序的UI，而流程控制则将孤立的模型和孤立的视图串联起来形成有机的系统。而在Java Web开发中，MVC更被诠释得淋漓尽致。在整个应用系统中，程序员可以针对数据库表或者概念模型，构造数据业务模型，编写适宜的Java分类，利用JSP来最终展现程序的功能以及接收用户请求。

现在，在Java Web开发中已经有流行的MVC开发框架，如Struts、Spring等，也有专注于Model层的Hibernate，还有专注于View和Controller之间交互的Webwork。

2.2 安装和配置JDK

万丈高楼平地起。不管是使用何种工具开发JSP，都需要用到Java虚拟机。为了搭建Java开发平台，首先必须安装JDK(J2SE Software Development Kit Standard Edition)，即Java2软件开发工具包标准版。这是Sun公司免费提供的Java开发工具。

2.2.1 安装JDK

本书以JDK-1.6.0_16为例介绍JDK的安装。该安装文件在本书配套素材中(文件路径：开发环境与工具\JDK安装)，文件名为jdk-6u16-windows-i586.exe，读者可以在Sun公司的网站http://java.sun.com上免费下载。

下面是安装JDK的步骤：

(1) 启动安装程序。双击安装文件jdk-6u16-windows-i586.exe，弹出如图2-2所示的对话框，单击“接受”按钮接受安装协议。

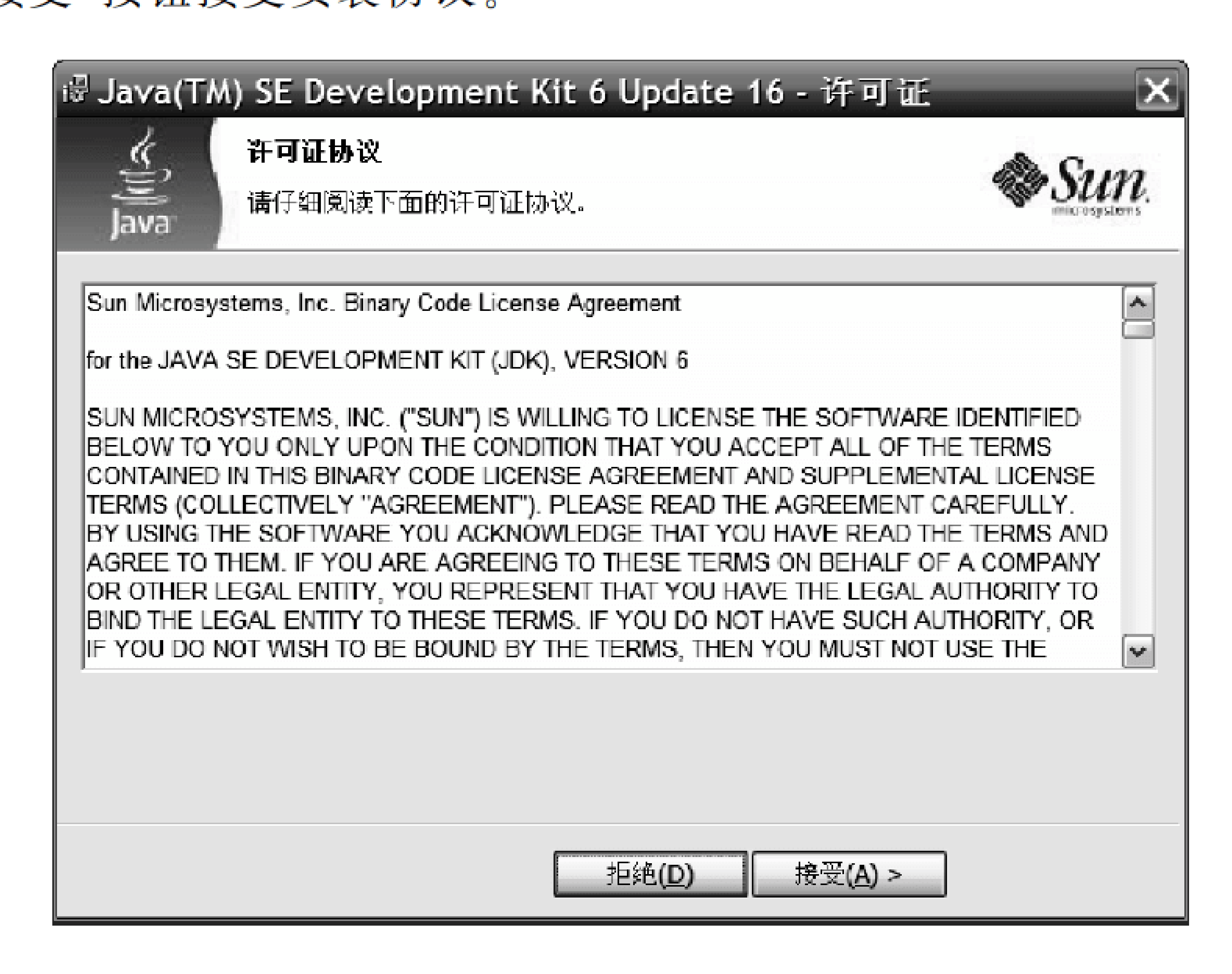


图2-2 执行jdk-6u16-windows-i586.exe

(2) 选择安装路径及安装内容。接受许可协议后，进入“自定义安装”面板，如图2-3所示。为了运行方便，有时需要改变安装路径，单击“更改”按钮即可选择安装路径。



图 2-3 “自定义安装”面板

(3) 执行安装。单击“下一步”按钮，出现“安装进度”面板，开始执行安装程序，安装成功后出现如图 2-4 所示的界面，单击“完成”按钮，完成 JDK 的安装。



图 2-4 安装成功

2.2.2 配置环境变量

安装完 JDK 后，需要配置环境变量。配置环境变量目的有 3 个：

(1) 让操作系统自动查找编译器、解释器所在的路径。

(2) 设置程序编译和执行时需要的类路径。

(3) Tomcat 服务器安装时需要知道虚拟机所在的路径。

配置环境变量的操作步骤如下。

(1) 在桌面上右击“我的电脑”图标，在弹出的快捷菜单中选择“属性”命令，弹出“系统属性”对话框。在“系统属性”对话框中选择“高级”选项卡，如图 2-5 所示。

(2) 在“高级”选项卡中单击“环境变量”按钮，弹出“环境变量”对话框，如图 2-6 所示。



图 2-5　“系统属性”对话框



图 2-6　“环境变量”对话框

(3) 在“系统变量”列表框中选择 Path 选项，单击“编辑”按钮或双击 Path 选项，打开“编辑系统变量”对话框，如图 2-7 所示。



图 2-7　“编辑系统变量”对话框

(4) 在“变量值”文本框中，将光标移动到现有文本的最后，先输入一个分号，然后再输入 C:\Program Files\Java\jdk1.6.0_16\bin，即安装 Java 后 bin 文件夹的完整路径。

(5) 单击所有打开对话框中的“确定”按钮，退出“系统属性”对话框，完成环境变量的配置。

2.3　安装和配置 Web 服务器

Web 服务器是 JSP 网页运行不可缺少的支撑平台，它的主要功能是对客户的请求进行处理和响应。Web 服务器有多种，其中 Tomcat 服务器是 Sun 公司在 JSWDK(JavaServer Web Development Kit)的基础上发展而来的一个优秀的 Web 服务器，它是由 JavaSoft 和 Apache 开发团队共同开发的产品。Tomcat 服务器自带 JSP 引擎和 Servlet 引擎。

2.3.1　安装 Tomcat

本书选用的 Web 服务器是 Tomcat 6.0.16。它是 Tomcat 一个较新的版本，读者可以从 http://tomcat.apache.org/index.html 上免费下载，或从本书配套素材中复制(文件路径：开发环境与工具\Tomcat 安装)，文件名为 apache-tomcat-6.0.16.exe。

安装 Tomcat 的操作步骤如下。

（1）双击 Tomcat 安装文件 apache-tomcat-6.0.16.exe，弹出 Apache Tomcat Setup 对话框，如图 2-8 所示。

（2）单击 Next 按钮，进入下一个 Apache Tomcat Setup 对话框，接受 Tomcat 使用协议，如图 2-9 所示。

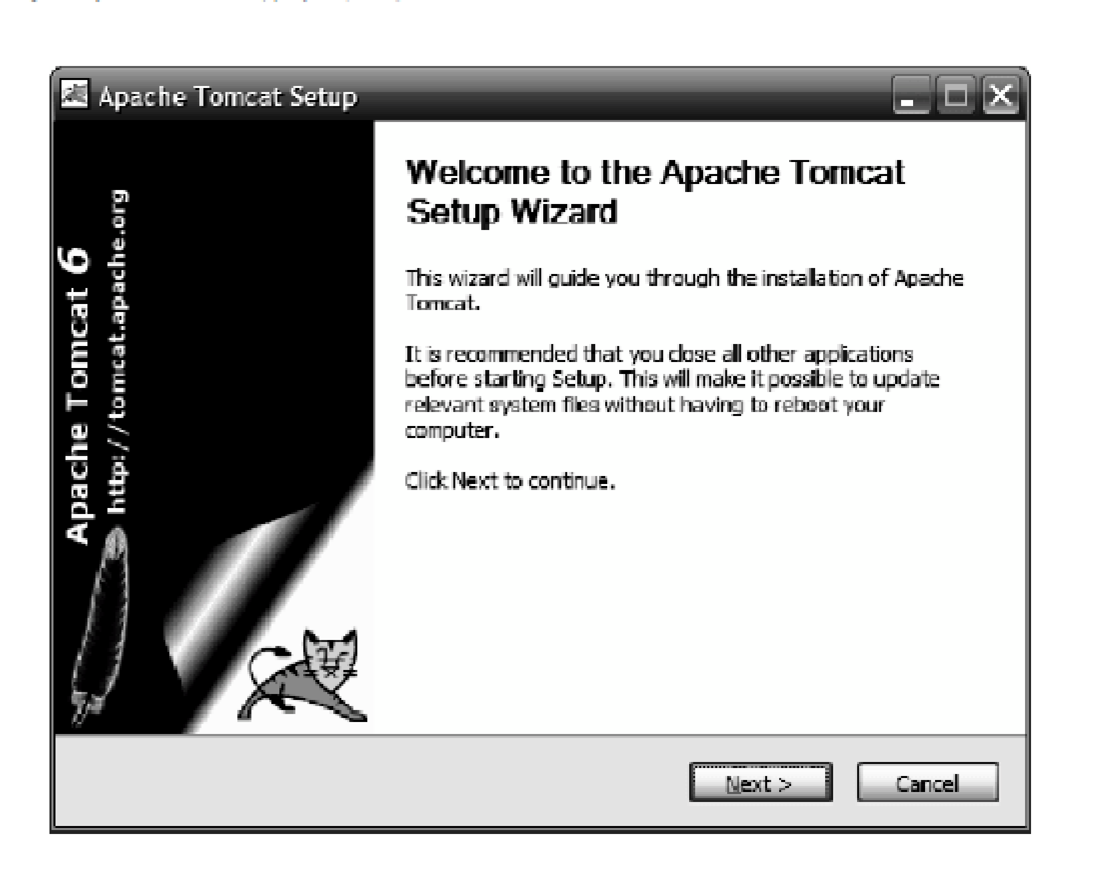


图 2-8　安装 Tomcat

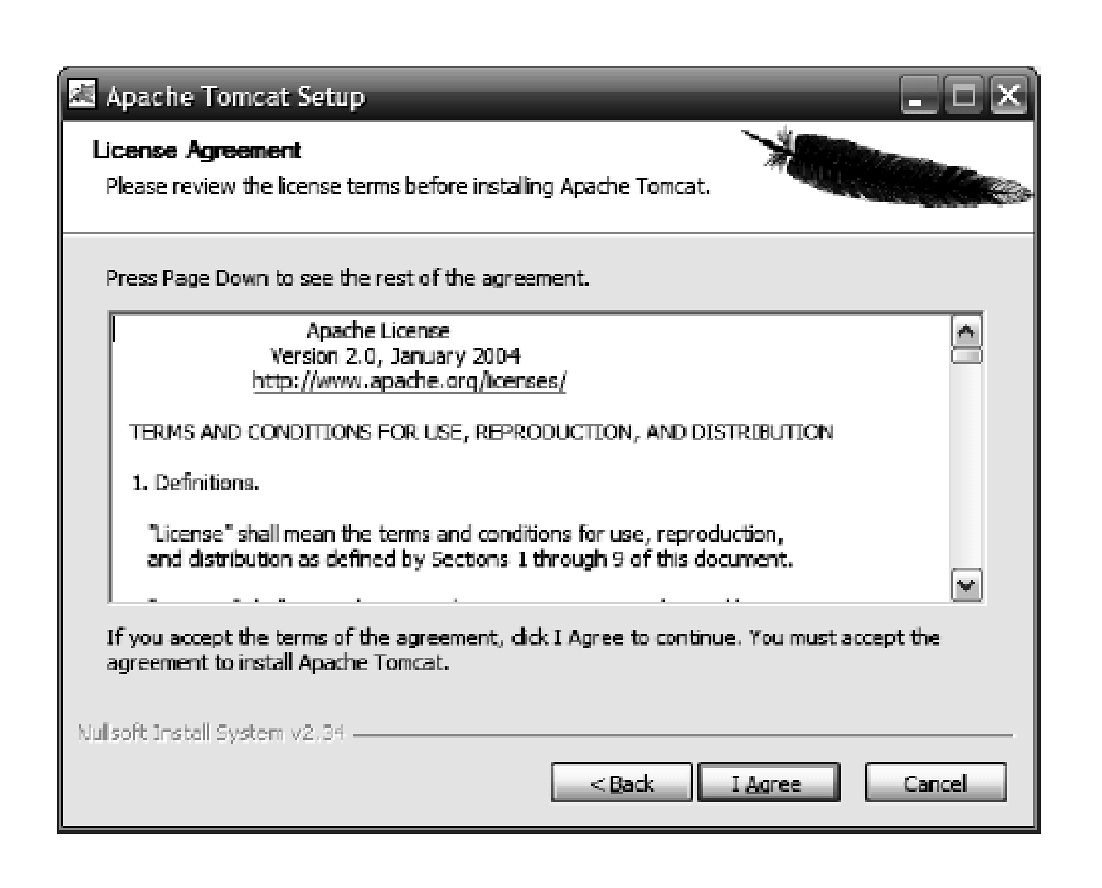


图 2-9　接受 Tomcat 使用协议

（3）单击 I Agree 按钮，再进入下一个 Apache Tomcat Setup 对话框，设置 Tomcat 的安装类型，如图 2-10 所示。

（4）在该页面的下拉列表框中选择“Full（完全安装）”选项，单击 Next 按钮，进入下一个 Apache Tomcat Setup 对话框，设定 Tomcat 的安装路径，如图 2-11 所示。单击 Browse 按钮，可选择安装路径。

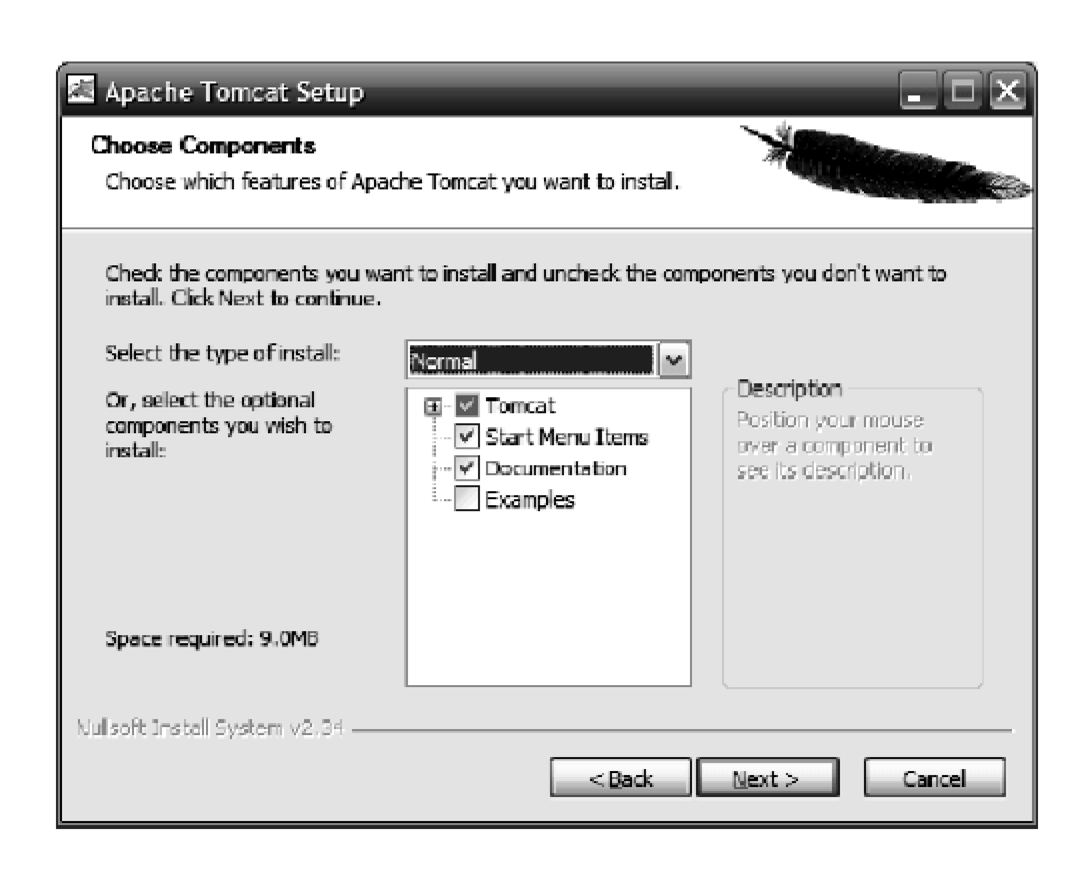


图 2-10　设置 Tomcat 的安装类型

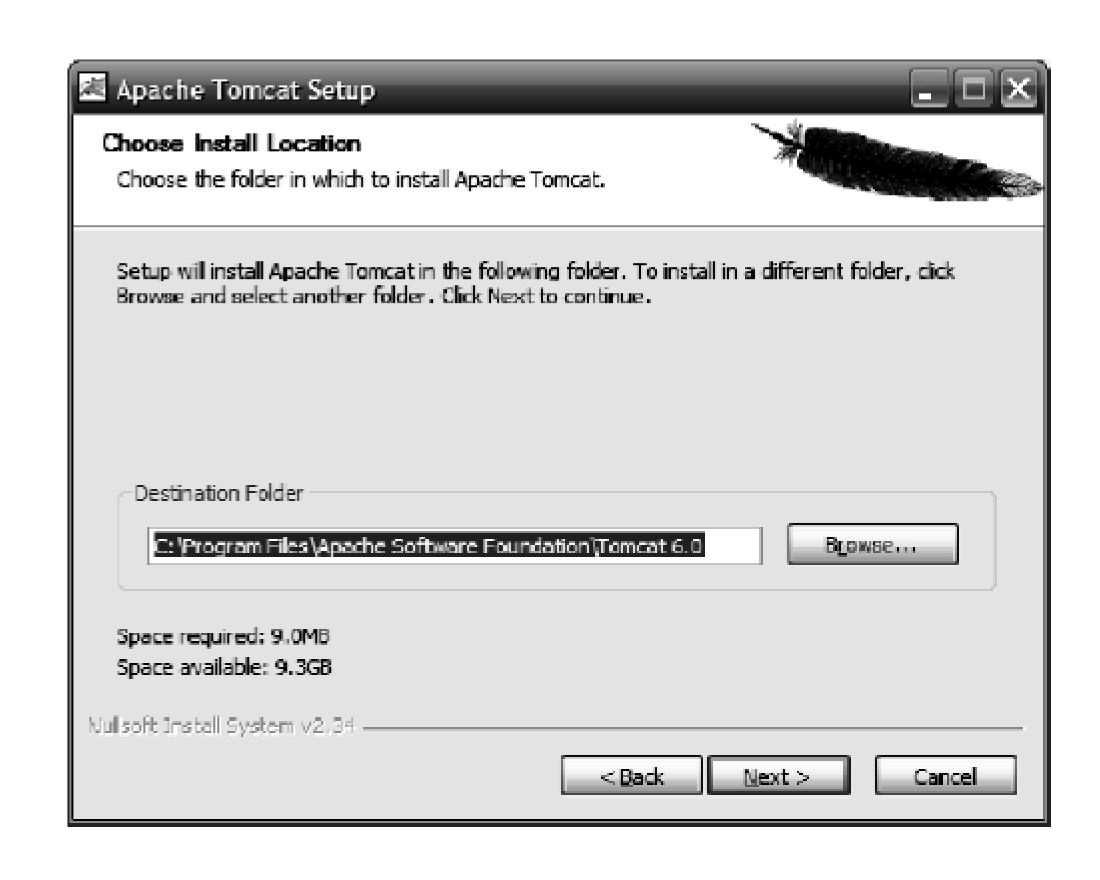


图 2-11　设置 Tomcat 的安装路径

（5）单击 Next 按钮，进入下一个 Apache Tomcat Setup 对话框，在该对话框中，可以设定 Tomcat 服务端口号、用户名和密码，在这里采用默认端口号和用户名，密码为空，如图 2-12 所示。

（6）单击 Next 按钮，进入下一个对话框，在该对话框中，单击文本框右边的“…”按钮，可以设置 Tomcat 所使用的 JVM（Java Virtual Machine）所在的路径，如图 2-13 所示。

（7）单击 Install 按钮，进入 Tomcat 文件的安装界面，如图 2-14 所示。

（8）文件配置完成后，将弹出如图 2-15 所示的 Apache Tomcat Setup 对话框，单击 Finish 按钮，完成 Tomcat 的安装。

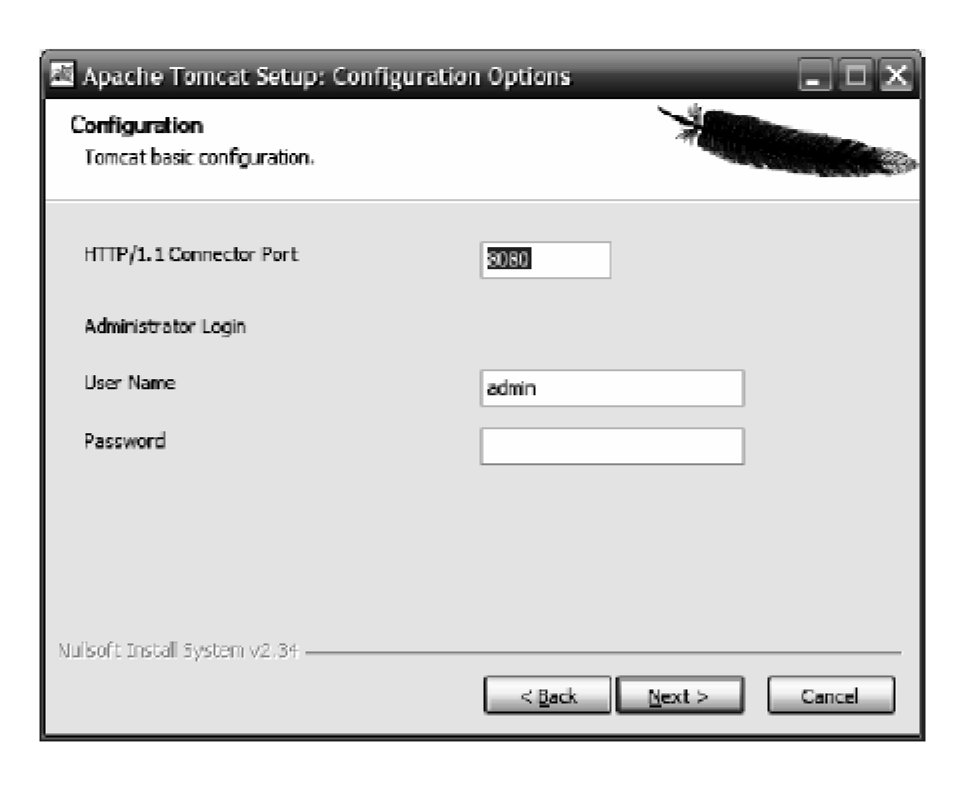


图 2-12　设置 Tomcat 的端口号、用户名和密码

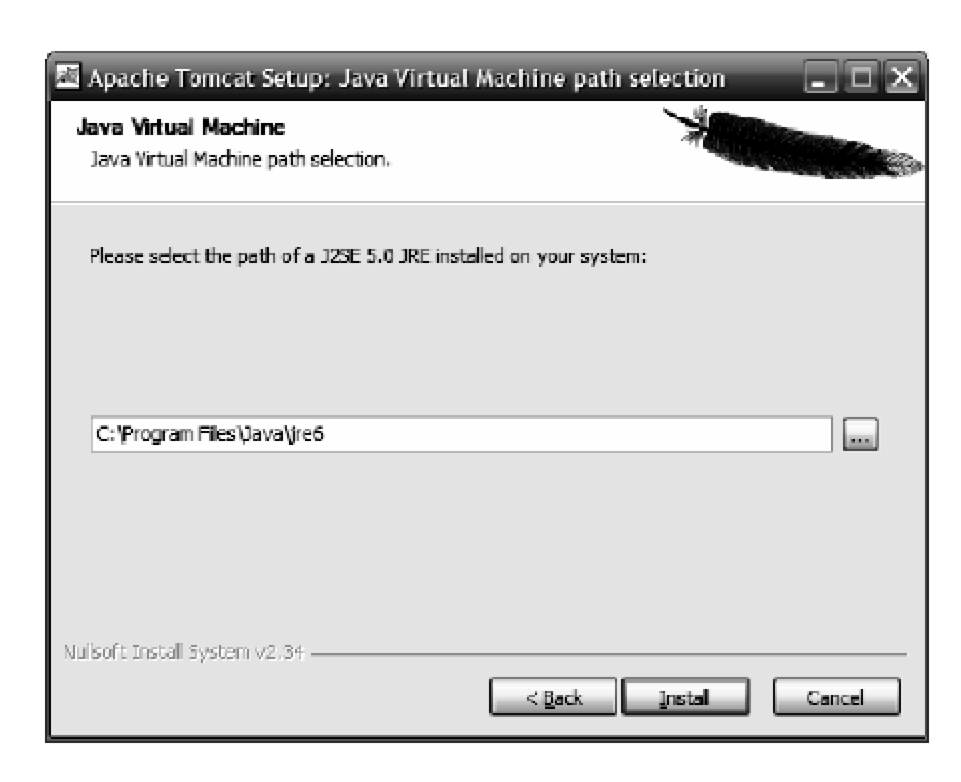


图 2-13　设置 Tomcat 使用的 JVM 路径

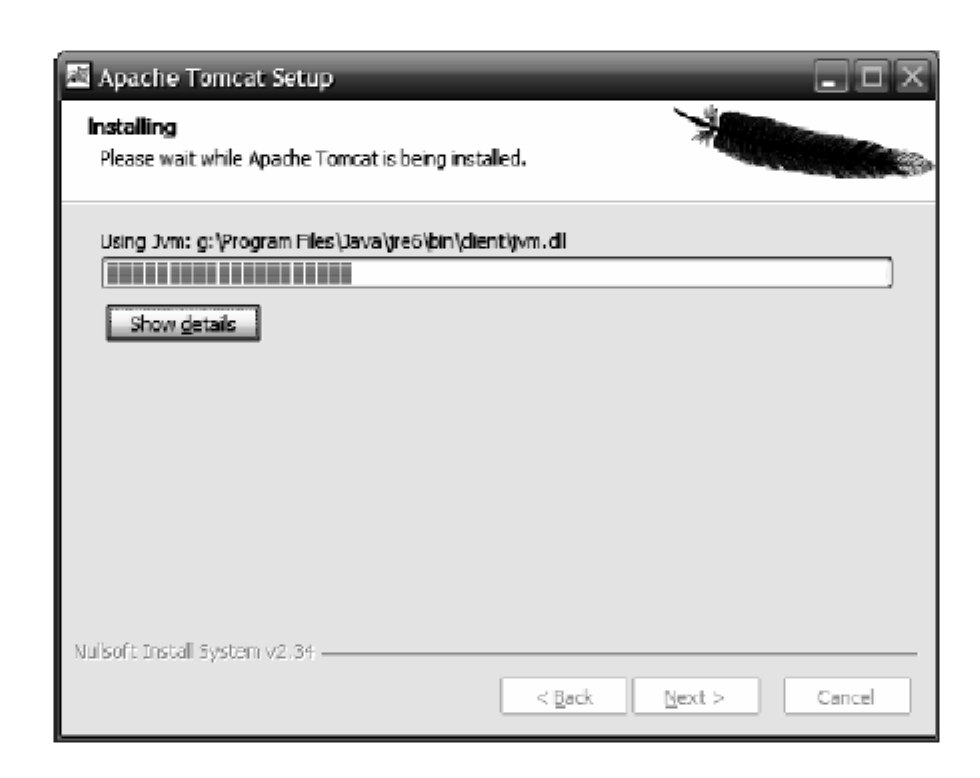


图 2-14　Tomcat 的安装界面

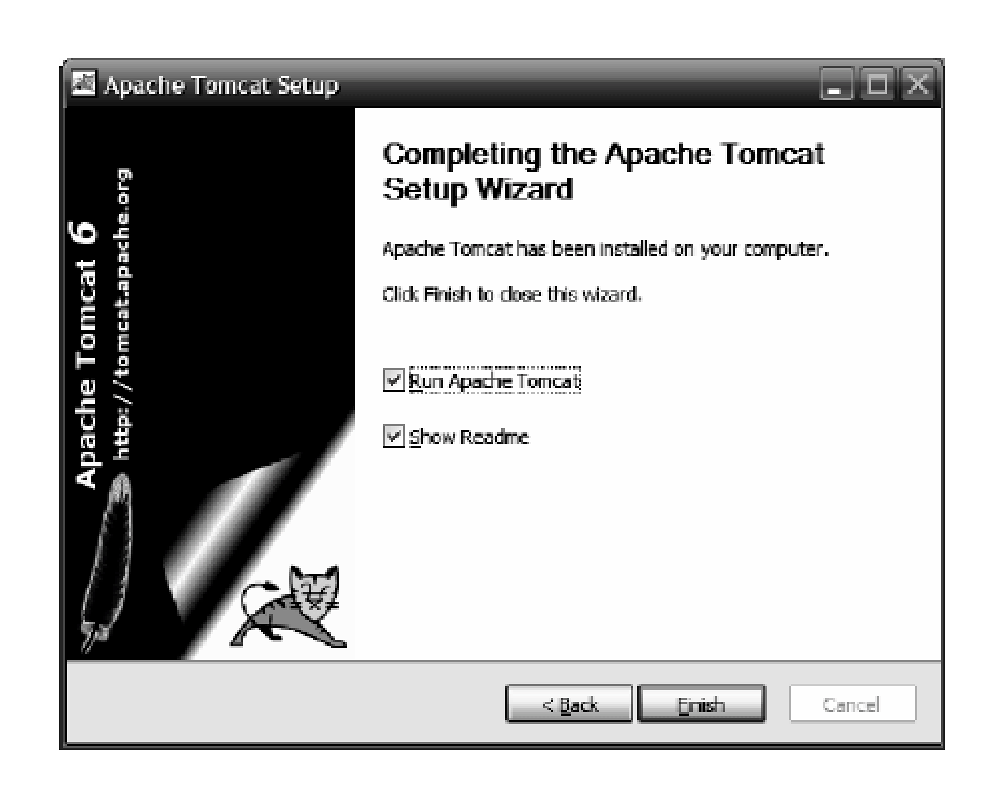


图 2-15　Tomcat 安装完成

（9）选择“开始”|“程序”|Apache Tomcat 6.0|Configure Tomcat 命令，弹出如图 2-16 所示的对话框，单击 Start 按钮，启动 Tomcat 服务器。

安装完 Tomcat 后，测试 Tomcat 是否安装成功。打开 IE 浏览器，在地址栏内输入 http://localhost:8080 或 http://127.0.0.1:8080。其中 localhost 或 127.0.0.1 表示本地主机，8080 表示访问的 Tomcat 服务器的端口号，http://localhost:8080 表示通过 8080 端口号访问本地主机上的 Tomcat 服务器。如果显示如图 2-17 所示的页面，表示 Tomcat 安装成功，否则需要重新安装。

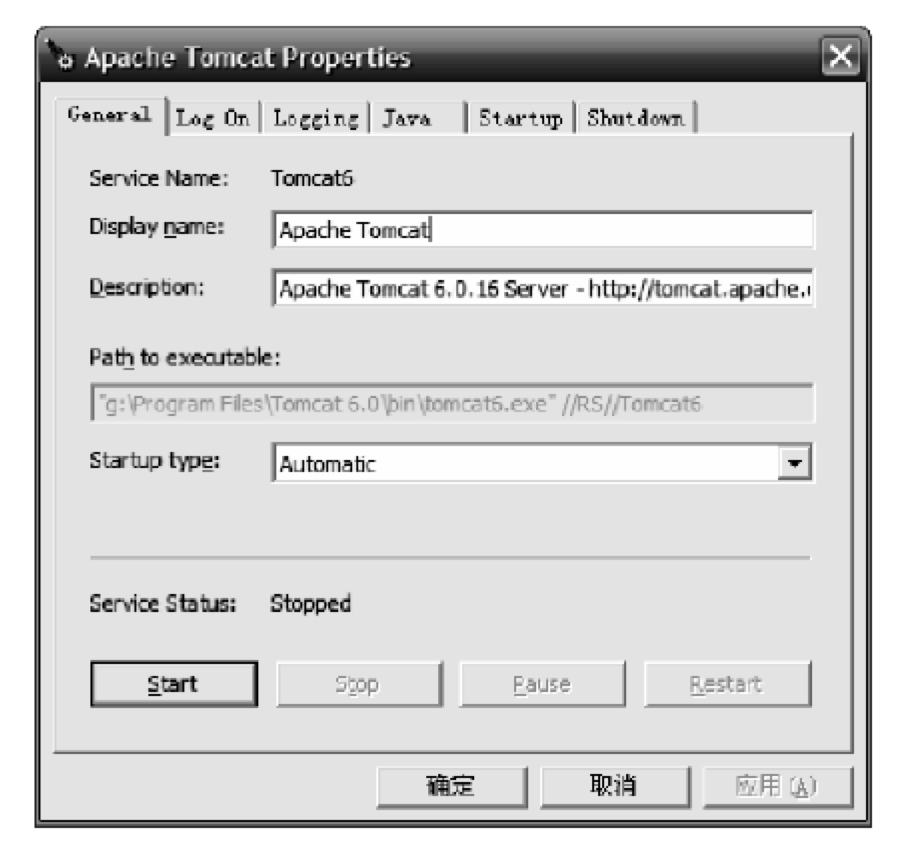


图 2-16　启动 Tomcat

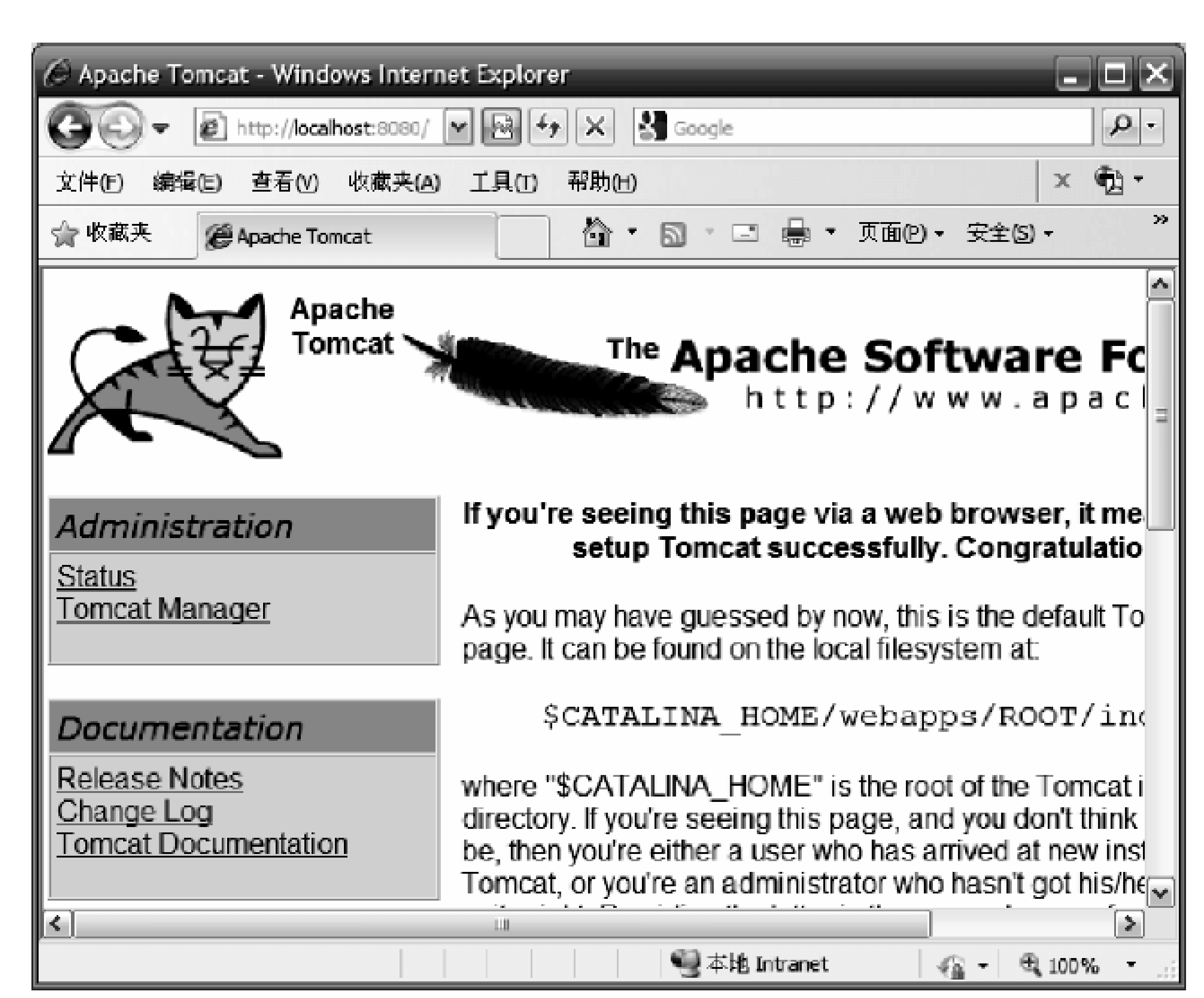


图 2-17　Tomcat 测试页面

2.3.2 Tomcat 的目录结构

在编写 JSP 页面前，首先需要了解 Tomcat 的目录结构和作用。Tomcat 安装完毕之后，安装目录下有若干目录，其目录结构如表 2-1 所示。

表 2-1 Tomcat 的目录结构

目录名	作 用
\bin	存放启动和关闭 Tomcat 服务器的文件
\lib	该目录下存放的 JAR 文件和类文件，能被各目录下的 JSP 页面和 Tomcat 服务器系统程序访问
\conf	存放服务器的各种配置文件，包括 server. xml、web. xml 等
\logs	存放服务器日志文件
\temp	存放 Tomcat 服务器的各种临时文件
\webapps	存放 Web 应用文件。如 JSP 应用例子程序、Servlet 应用例子程序和默认 Web 服务目录 ROOT
\work	存放 JSP 页面转换为 Servlet 文件和字节码文件

2.3.3 配置和测试 Tomcat

1. Tomcat 默认的 Web 服务目录

从 Tomcat 的目录结构可以看出，Tomcat 服务器的默认 Web 服务目录是\Tomcat 6.0\webapps\ROOT，用户开发的 JSP 页面程序需要保存在该目录下，Tomcat 已经自动配置好了其他选项，可直接运行。

例如，用记事本编辑一个页面程序 example. jsp，源代码如下：

```
                    <%-- 例程 2-1example.jsp --%>
<%
    out.print(" My first example program  !") ;
%>
```

将 example. jsp 程序保存到\Tomcat 5.0\webapps\ROOT 目录下，在浏览器地址栏中输入以下网址：http://localhost:8080/example. jsp，运行结果如图 2-18 所示。

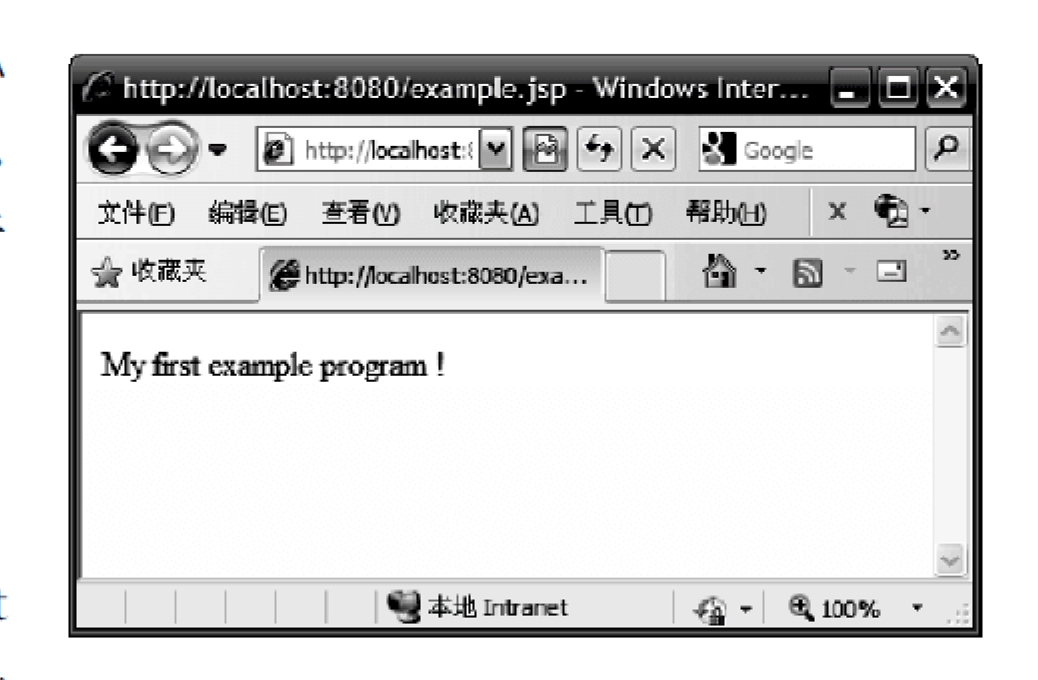


图 2-18 测试默认的 Web 目录

2. 建立自己的 Web 目录

开发人员可以将 JSP 页面程序部署在 Tomcat 服务器的默认 Web 目录下，也可以部署在自己创建的 Web 目录下。下面是创建 Web 目录的步骤：

（1）在服务器上创建自己的目录，如创建一个目录为 E:\code\2。

（2）配置 Web 目录。用记事本打开 \Tomcat 6.0\conf 目录下的文件 server. xml，在该文件末尾有一个标识符</Host>，在该标识符前面添加以下语句：

```
<Context   path="/2"    docBase="E:/code/2"     debug="0"    reloadable="true">
</Context>
```

该语句的作用是将目录 E:/code/2 设置为 Web 目录，将该目录下 JSP 页面程序的执行路径设置为/2。属性 docBase 的值为 E:/code/2 指定 Web 目录的物理路径，属性 path 的值为/2，它指定执行 E：/code/2 目录下 JSP 页面程序时的路径(可自行设定)。

现在，建立了自己的 Web 目录，并修改和保存了 server.xml 文件。将 example.jsp 页面程序复制到 E:/code/2 目录下面，重新启动 Tomcat 服务器，然后在浏览器地址栏中输入以下网址：http://localhost:8080/2/example.jsp，运行结果如图 2-19 所示。

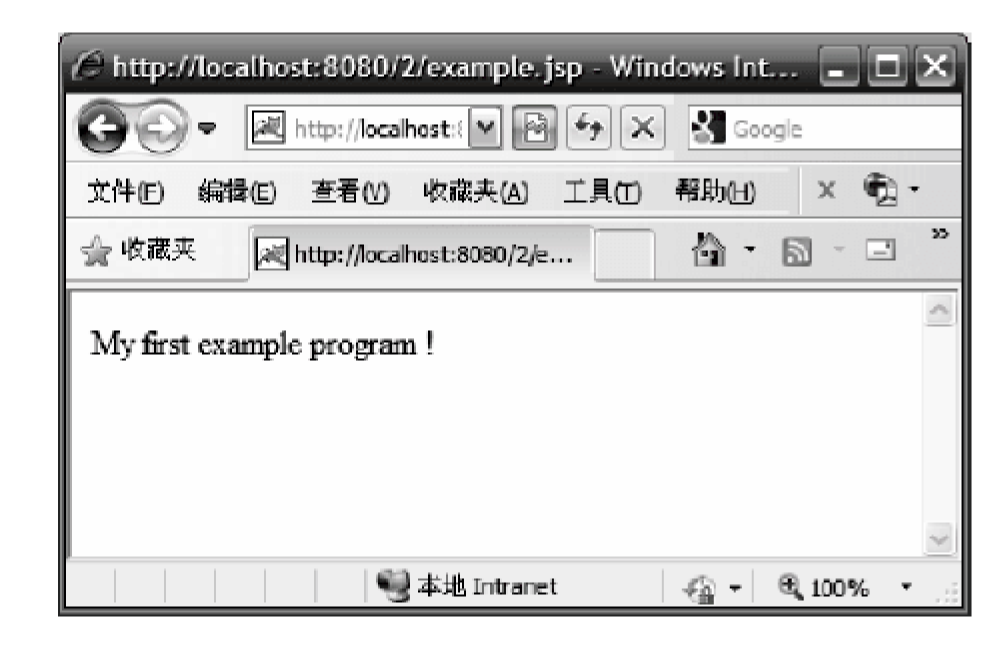


图 2-19　测试创建的 Web 目录

为了方便操作，本书中创建了 9 个 Web 服务目录，这 9 个目录名分别是 E:\code\2、E:\code\3、E:\code\4、E:\code\5、E:\code\6、E:\code\7、E:\code\8、E:\code\9 和 E:\code\10，用为分别保存各章的 JSP 页面程序。

打开 server.xml 文件，在标识符</Host>前面添加下面的语句：

```
<Context   path="/2" docBase="E:/code/2" debug="0" reloadable="true">
</Context>

<Context   path="/3" docBase="E:/code/3" debug="0" reloadable="true">
</Context>

<Context   path="/4" docBase="E:/code/4" debug="0" reloadable="true">
</Context>

<Context   path="/5" docBase="E:/code/5" debug="0" reloadable="true">
</Context>

<Context   path="/6" docBase="E:/code/6" debug="0" reloadable="true">
</Context>

<Context   path="/7" docBase="E:/code/7" debug="0" reloadable="true">
</Context>

<Context   path="/8" docBase="E:/code/8" debug="0" reloadable="true">
</Context>

<Context   path="/9" docBase="E:/code/9" debug="0" reloadable="true">
</Context>

<Context   path="/10" docBase="E:/code/10" debug="0" reloadable="true">
</Context>
```

保存修改后的 server.xml 文件，并重新启动 Tomcat。

专家点拨：server.xml 文件修改后，必须重新启动 Tomcat 服务器，才能使修改生效。

2.3.4 JSP 页面的执行流程

一般情况下，一个 JSP 页面会有多个客户访问，当第一个客户访问 JSP 页面时，JSP 页面的执行过程如下所述。

(1) 客户通过浏览器向服务器端的 JSP 页面发送请求。

(2) JSP 引擎检查 JSP 文件对应的 Servlet 源代码是否存在，若不存在转向第(4)步，否则执行下一步。

(3) JSP 引擎检查 JSP 页面是否修改，若未修改，转向第(5)步，否则执行下一步。

(4) JSP 引擎将 JSP 页面文件转译为 Servlet 源代码(相应的.java 代码)。

(5) JSP 引擎将 Servlet 源代码编译为相应的字节码(.class 代码)。

(6) JSP 引擎加载字节码到内存中。

(7) 字节码处理客户请求，并将结果返回给客户。

在不修改 JSP 页面的情况下，除了第一个客户访问 JSP 页面需要经过以上几个步骤外，以后访问该 JSP 页面的客户请求，会直接被发送给 JSP 对应的字节码程序进行处理，并将处理结果返回给客户。在这种情况下，JSP 页面既不需转译也不需编译，JSP 页面执行效率非常高。

专家点拨：要使修改后的 JSP 页面有效，必须重新启动服务器，以便重新加载修改后的 JSP 页面。

2.4 上机指导

2.4.1 安装 JDK 和 Tomcat

1. 练习目标

(1) 熟悉 JDK 开发包的安装。

(2) 熟悉 Tomcat 服务器的安装。

2. 练习指导

(1) 下载或者从本书的配套素材中复制 JDK 和 Tomcat 工具。

(2) 按照前面 2.2.1 节及 2.3.1 节的操作步骤安装 JDK 工具和 Tomcat 工具。

2.4.2 配置 JSP 的运行环境

1. 练习目标

(1) 熟悉 JSP 运行环境的配置。

(2) 熟练创建自己的运行环境。

2. 练习指导

(1) 右键单击桌面上的"我的电脑"图标,在弹出的快捷菜单中选择"属性"命令,打开"系统属性"对话框,或者在"控制面板"中双击"系统"图标,也可打开"系统属性"对话框,选择"高级"选项卡。

(2) 单击"环境变量"按钮,打开"环境变量"对话框,在"系统变量"列表中选中 Path 选项,单击"编辑"按钮,打开"编辑系统变量"对话框。

(3) 在"变量值"的文本框中将光标放在最后,输入一个分号";",然后再输入 JDK 的安装路径。

(4) 单击"确定"按钮完成 JDK 环境的配置。

(5) 默认情况下,Tomcat 不需要进行配置,但实际应用中,为了操作方便,要创建自己的服务器目录,读者可参考 2.3.3 节的方法来建立自己的 Web 目录。

2.4.3 计算 1+2+3+…+100 的和并输出当时的日期和时间

1. 练习目标

(1) 熟悉创建 JSP 页面的方法。

(2) 了解 JSP 程序的代码。

2. 练习指导

(1) 首先定义一个求和的方法,如使用 for 循环语句。

(2) 调用 Java 的 Date()方法显示当时日期。

(3) 通过表达式输出相加的结果。

本程序的源代码如下:

```
<%-- 例程 2-2 sum.jsp --%>

<%@page contentType="text/html; charset=GBK"%>
<html>
<head>
<title>一个简单的 JSP 应用</title>
</head>
<body>
<%!int i, total;%>
<%
     for (i = 1, total = 0; i <= 100; i++)
       {
           total = total + i;
       }
 %>
<p>  从 1 加到 100 的结果是:<% = total%></p>
<p><% = new java.util.Date()%></p>
</body>
</html>
```

该程序源代码为本书配套素材上的 sum.jsp 文件(文件路径为 code\2\上机指导)。

本章小结

本章首先介绍开发 JSP 的使用工具,然后着重讲解开发 JSP 的必备工具 JDK 开发包以及 Tomcat 服务器的安装与配置,随后还介绍了 Web 默认目录和新建 Web 服务目录的步骤,主要为后面的学习打下基础。

习题 2

一、简答题

1. 简述目前比较流行的 JSP 开发工具。
2. 如何配置 JDK 环境?
3. 如何创建自己的 Web 目录?
4. 简述 JSP 页面的执行流程。

二、操作编程题

1. 安装 JDK。
2. 安装并配置 Tomcat。
3. 建立一个文件夹 test,并将其设置为 Web 服务目录。
4. 编写一个显示当前时间的 JSP 页面程序,并运行该程序。

第3章 JSP语言基础

JSP是一种基于Java Servlet的Web开发技术，它是一种功能强大、可以实现跨平台操作的动态网页开发技术。它使得构造基于Web的应用程序更加容易和快捷，而这些应用程序能够与各种Web服务器、应用服务器、浏览器和开发工具共同工作，目前已经成为开发动态网页的主流技术之一。本章主要介绍JSP页面组成元素的语法及使用方法。

本章主要内容：

- HTML标记的语法、作用和使用方法；
- JavaScript脚本语言的应用；
- JSP的基本语法；
- JSP指令标签和动作标签的使用方法。

3.1 HTML基础知识

超文本标记语言（Hypertext Marked Language，HTML）是一种用来描述超文本文档的标记语言，用HTML编写的超文本文档能独立于各种操作系统平台（如UNIX、Windows等），JSP就是在HTML中嵌入Java脚本。

3.1.1 HTML文档结构

HTML文档是在普通文件中的文本上加上标签，使其达到预期的显示效果。当浏览器打开一个HTML文档时，会根据标签的含义显示HTML文档中的文本。其中标签由“<标签名称 属性>”来表示。

1. HTML标签的结构形态

1）<标签> 元素 </标签>

标签的作用范围从<标签>开始，到</标签>结束。例如，<h2>demo</h2>，其作用就是将demo这段文本按<h2>标签规定的含义来显示，即以2号标题来显示。而<h2>和</h2>之外的文本不受这组标签的影响。

2）<标签 属性名 = "属性值"> 元素 </标签>

其中属性往往表示标签的一些附加信息，一个标签可以包含多个属性，各属性之间无先后次序，用空格分开。例如，<body background="back_ground.gif" text="red">hello

</body>，其中 Background 属性用来表示 HTML 文档的背景图片，text 属性用来表示文本的颜色。

3）<标签>

标签单独出现，只有开始标签而没有结束标签，也称为“空标签”。

2. HTML 文档结构

HTML 文档分“文件头”和“文件体”两部分，在文件头里，对这个文档进行了一些必要的定义，文件体中的信息才是要显示的各种文档信息，HTML 文档的结构如下所示。

```
<html>
  <head>
    头部信息，如标题
  </head>
  <body>
    在这里放置网页的内容，包括文本、超链接、图像、动画等
  </body>
</html>
```

其中<html>在最外层，表示这对标签间的内容是 HTML 文档。一些 HTML 文档省略了<html>标签，因为扩展名为 html 或 htm 的文件被 Web 浏览器默认为是 HTML 文档。<head>与</head>之间包括文档的头部信息，如文档的标题等，若不需要头部信息则可省略此标签。<body>标签一般不省略，表示正文内容的开始。

【例 3-1】 显示欢迎光临主页。

本例是显示一个简单的超文本文档，使用 HTML 的一些常用标签，如标题、字体等。

```
<!-- 例程 3-1 first.html -->
<html>
<head>
<title>一个简单的 HTML 文档</title>
</head>
<body>
    <h1>欢迎光临</h1>
    <br>
    <font size="5"  face="华文行楷"  color="red">
          这是我的第一个主页，欢迎大家的访问！
    </font>
</body>
</html>
```

该代码输出结果页面如图 3-1 所示。本例源代码存放于本书配套素材中的 first.html 文件中（文件路径为 code\3）。

专家点拨：关于 HTML 的常用标签，请读者参考有关 HTML 语言的书籍，本书不作详述。

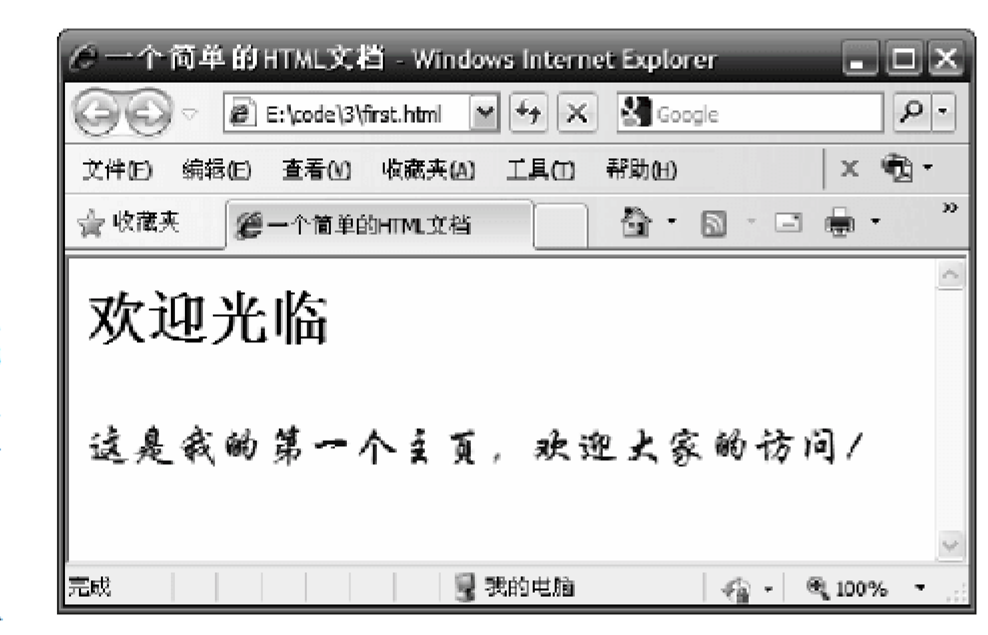


图 3-1　first.html 的输出结果

3.1.2 HTML 表单

在网页设计中，一般使用 HTML 标签创建用户界面，实现输入数据和展示数据。网页中这种由可输入表项及项目选择等元素所组成的栏目称为“表单”，使用表单可以实现页面的数据传送，还可以实现 Web 程序和用户的交互。

表单通常和程序连接(如 JSP 程序)来实现数据的处理，一个表单有 3 个基本组成部分：

(1) 表单标签：包含了处理表单数据所用程序的位置及数据提交到服务器使用的方法。

(2) 表单组件：用来输入数据或进行选择，包括单行文本编辑框、密码框、单选按钮、复选框、多行文本编辑框、下拉列表框等。

(3) 表单按钮：包括“提交”按钮和“重置”按钮，用来提交输入或者取消输入。

1. 表单

表单本身是一个框架，它把提交组件、数据输入组件和格式化组件组合在一起，构成一个用户输入界面，其作用是利用提交组件将表单中的数据(数据输入组件接受数据)提交给服务器。表单的基本语法如下：

```
<form method = get/post action = "accept.jsp" name = "表单名字">
      [数据输入组件(1 至多个组件)][ 格式化组件]
      提交组件 [重置组件]
</form>
```

【说明】

(1) <form>是表单开始标记，</form>是表单结束标记，在起始和结束标记之间可以放置多个数据组件，以接收用户输入的数据。还可以放置一个提交组件，一个重置组件。所有表单组件都应该在<form>和</form>标签之内。

(2) </form> method 属性表示提交信息的方式，用 get 方法提交时，信息会显示在浏览器的地址栏中，而用 post 方法提交时，信息不会显示在地址栏中。

(3) action 的值是一个文件，该文件接收表单数据，即把表单数据提交给该文件。

(4) 提交组件：当用户单击该组件时，表单中的数据组件值被传递给页面 accept.jsp。

专家点拨：一个页面可以有多个表单 form，但两个表单不可以嵌套或者重叠。

2. 单行文本编辑框

一般而言，用户通过文本框输入各种数据。文本框的一般语法格式如下：

```
<input type = "text"
        name = "textname"
        value = "defaultvalue"
        size = "lengthvalue"
        align = "left"/"center"/"right"
        maxlength = "inputvalue">
```

【说明】

(1) type 的值为 text 时，表示这个组件的类型是单行文本编辑框。

(2) name 的值是为文本框指定的名字。
(3) value 的值是文本框的初始值。
(4) size 的值是文本框宽度,单位是字符。
(5) align 的值是文本框在浏览器中的对齐方式。
(6) maxlength 的值指定文本框可输入字符的最大长度。

3. 密码框

密码框是一种特殊的文本框,输入的信息看不到,用 * 号回显,防止他人偷看密码。密码框的一般语法格式如下:

```
< input type = "password"
        name = "passwordname"
        size = "lengthvalue"
        align = "left"/"center"/"right"
        maxlength = "inputvalue">
```

【说明】
(1) type 的值为 password 时,表示本组件是密码框,它与单行文本框的区别在于用户输入的内容不被显示。
(2) name 的值是为密码框指定的名字。

4. 单选按钮

当一个题目中的答案只能多选一时,就使用单选按钮,系统给出几种选择,用户从中选择一项。单选按钮的一般语法格式如下:

```
< input type = "radio"
        name = "radioname"
        value = "radiovalue"
        checked = "str">
```

【说明】
(1) type 的值为 radio 时,表示本组件是单选按钮。
(2) name 的值是为单选按钮指定的名字,同一组单选按钮的名字应相同。当一个单选按钮被选中时,服务器可以通过 name 的值获得被选中的单选按钮的 value 值。
(3) checked 的值如果是一个非空的字符串,则表示该单选按钮默认被选中,同一组单选按钮同时只能有一个被选中。

5. 复选框

在给出的选项中可以同时选择多个选项。当在一个题目中可以选择多个答案时,就使用复选框。复选框的一般语法格式如下:

```
< input type = "checkbox"
        name = "checkboxname"
        value = "checkvalue"
```

```
        align = "top"/"bottom"
        checked = "str" >
```

【说明】

(1) type 的值为 checkbox 时,表示本组件的类型是复选框。

(2) name 的值是为复选框指定的名字。在同一组复选框中,每个复选框 name 的值应相同。当复选框被选中时,服务器可以通过 name 的值获得 value 的值。

(3) checked 的值如果是一个非空的字符串,那么该复选框的初始状态就是选中状态。同一组复选框可以同时有多个被选中。

6. 列表框

列表框的功能是用户在给出的较多选项中选择一个。下拉式列表和滚动式列表框都是通过<select>和<option>标记来定义的。列表框的基本格式为:

```
<select name = "listname" size = "showrows">
<option value = "value1">
<option value = "value2">
 ⋮
<option value = "valuen" selected >
</select>
```

【说明】

(1) 一个列表框由一个<select>和若干个<option>组成,<option>标签表示列表框中的一个选项。

(2) name 的值指定列表框的名字,size 的值是 1 时,是下拉列表;大于 1 时,则是滚动列表框。size 的默认值是 1。

(3) 服务器通过 name 的值获得列表框被选中项的值。value1、value2……表示各选项的值,而 selected 则表示这一选项在默认状态是被选中的项。

7. 多行文本编辑框

该组件在表单中指定一个能够进行多行文本输入的文本区,其语法格式如下:

```
<textarea   name = "textareaname"
            rows = "showrows"
            cols = "showcols">
```

【说明】

(1) name 的值指定文本区的名字。

(2) rows 的值指定文本区的可视高度。

(3) cols 的值指定文本区的可视宽度。

8. 表格

表格经常用于对显示信息和输入信息的格式进行排版。表格的基本语法如下:

```
<table>
  <tr><th> 表头字段 11 </th>…<th>表头字段 1n </th>  </tr>
```

```
  <tr><th> 表头字段 21 </th>…<th>表头字段 2n </th>  </tr>
   ⋮
  <tr><td> 数据 11  </td>…<td>  数据 1n  </td>  </tr>
  <tr><td> 数据 21  </td>…<td>  数据 2n  </td>  </tr>
   ⋮
</table>
```

【说明】

（1）一个表格由<table>标记开始，以</table>标记结束。

（2）一个表格由 1 至多个表头行和 1 至多个数据行组成。在<tr>和</tr>标记之间，定义一个表头行或一个数据行。

（3）一行包含多个单元格，单元格分为表头单元格和数据单元格两种。

（4）在<th>和</th>之间定义一个表头单元格，以显示表头字段；在<td>和</td>之间定义一个数据单元格，以显示数据项。

9. 提交按钮

当用户按下此按钮后，表单所包含的数据被提交到服务器。每个表单中都应该有至少一个提交按钮来完成提交动作。其语法格式如下：

```
<input type="submit"
       value="提交"
       name="buttonname">
```

【说明】

（1）type 的值是 submit，表明该组件是数据提交组件。

（2）value 的值是组件上的标识符。

（3）name 是按钮的名称。

10. 重置按钮

当用户需要重新填写表单中的数据时，按下此按钮则清除表单中各项数据。重置按钮的语法格式如下：

```
<input type="reset"
       value="清除"
       name="buttonname">
```

【说明】

（1）type 的值为 reset 时是重置按钮的标识。

（2）value 是重置按钮的值，同时也是按钮上面显示的内容。

专家点拨：重置按钮完成的功能是恢复页面的信息，但并不是所有的页面都需要重置按钮。

【例 3-2】 创建一用户界面，注册用户的姓名、性别、爱好等信息，实现表单提交。

本例用表单创建一个注册信息的用户界面，表单中包含了文本框、单选按钮、复选框、列表框、提交按钮等。

```
<!-- 表单例程 3-2 form.html -->

<form method="get" name="form1" action="formdeal.jsp">
  <h1 align=center><font face="华文新魏">注册信息</font></h1>
  <p>姓名:<input type="text" name="name" size="20">
          性别:<input type="radio" value="男" checked name="sex">男
                <input type="radio" name="sex" value="女">女
          职业:<select size="1" name="work">
                  <option>教师</option>
                  <option>公务员</option>
                  <option>军人</option>
                  <option>工人</option>
                  <option selected>职员
                  </option>
                </select>
  </p>
  <p>联系电话:<input type="text" name="tel">
      邮箱地址:<input type="text" name="email"><p>
  <p>爱好:
           <input type="checkbox" name="CC" value="旅游">旅游
           <input type="checkbox" name="CC" value="音乐" checked>音乐
           <input type="checkbox" name="CC" value="游戏">游戏
           <input type="checkbox" name="CC" value="看书">看书
           <input type="checkbox" name="CC" value="保龄">保龄
           <input type="checkbox" name="CC" value="看电影">看电影
  </p>
  <p>密码:<input type="password" name="pass" size="20">
      确认密码:<input type="password" name="confirm" size="20"></p>
  <p>自我描述:</p>
  <p align=center><textarea rows="8" name="des" cols="60"></textarea></p>
  <p align=center>
     <input type="button" value="提交" name="B1" onclick="check()">
     <input type="button" value="重置" name="B2" onclick="check()">
  </p>
</form>
```

该例程在网页上的显示输出如图 3-2 所示。本例源代码存放于本书配套素材中的 form.html 文件中(文件路径为 code\3)。

图 3-2　form.html 的输出

3.1.3 JavaScript 基础

JavaScript 是一种基于对象和事件驱动的脚本语言，常在 Web 开发中用于增强网页与应用程序间的交互，从而可以开发客户端的应用程序。JavaScript 是通过嵌入在标准的 HTML 文件中实现的，可以直接控制浏览器窗口中的各元素以及页面上的内容。JavaScript 的一个重要功能就是在 JSP 程序中实现对客户端输入的验证。

JavaScript 的特点如下。

（1）简单：JavaScript 是一种脚本编写语言，它的基本语法与 C 和 C++类似。

（2）动态：JavaScript 是动态的，它可以直接对用户输入做出响应，无须经过 Web 服务器验证。

（3）跨平台性：JavaScript 只依赖浏览器，与系统环境无关，只要能运行支持 JavaScript 的浏览器就可以正确执行 JavaScript 程序。

（4）基于事件：JavaScript 对用户的响应，是采用事件驱动的方式进行的。

1. JavaScript 中的事件

JavaScript 事件驱动中的事件是通过鼠标或热键的动作引发的，其主要有以下几个事件。

1）单击事件 onClick

当用户用鼠标单击某个对象时，产生 Onclick 事件，同时 Onclick 指定的事件处理程序代码将被调用执行。

2）改变事件 onChange

当文本输入区的内容被更改，或列表框中的某个选项状态被改变时引发该事件。

3）选中事件 onSelect

当文本组件中的文字被选中时引发该事件。

4）获得焦点事件 onFocus

当某个表单元素获得焦点时引发该事件。

2. JavaScript 的常用方法

JavaScript 能方便地使用浏览器环境提供的对象，这些对象主要包括 Windows 对象（窗口对象）、Document 对象（文档对象）、Location 对象（位置对象）及 History 对象（历史对象），利用这些对象，可以实现与 Web 页面的交互，其中 Windows 对象是所加载文档的对象，调用 Windows 对象的方法可以直接写方法名，而 Document 对象包含了与文档对象一起工作的对象。这两个对象的常用方法如下：

（1）window. alert()方法：创建一个具有 OK 按钮的信息框。

（2）window. confirm()方法：为编程人员提供一个具有两个按钮的对话框。

（3）window. prompt()方法：允许用户在对话框中输入信息。

（4）document. write()方法和 document. writeln()方法：用于将文本信息直接输出到浏览器窗口中。

3. 使用 JavaScript 实现客户端验证的常见语法

1）脚本标签

```
<script language = "javaScript">
</script>
```

在<script>和</script>之间编写 JavaScript 代码。

2）定义函数

```
function check()
 {
     //函数体
 }
```

3）获取表单元素 pass 的值

```
document.form1.pass.value
```

4）获取表单元素 pass 的长度

```
document.form1.pass.value.length
```

5）在表单元素 pass 上设置光标

```
document.form1.pass.focus()
```

6）在表单元素 pass 中查找字符

```
document.form1.pass.indexOf('@')
```

7）获取表单元素 pass 中的第 i 个字符

```
document.form1.pass.charAt(i)
```

8）将表单提交给 Web 服务器上的处理程序

```
document.form1.submit()
```

【例 3-3】 使用 JavaScript 进行客户端验证表单。

本例验证了名字是否为空、邮箱是否为空以及邮箱地址格式是否正确、确认密码与密码是否一致。

```
<!-- 例程 3-3 formtijiao.html -->

<html>
  <head>
    <meta http-equiv = "Content-Type" content = "text/html; charset = gb2312">
  </head>
  <script language = "javascript">
    function check()
     {
        if (document.form1.pass.value! = document.form1.confirm.value)
```

```
          {
              window.confirm("对不起,密码有误!");
              document.form1.pass.focus();
              return false;
          }
         if (document.form1.name.value == "")
          {
              window.alert("请输入姓名!");
              document.form1.name.focus();
          return false;
          }
         if (document.form1.email.value == "")
          {
              window.alert("请输入 E-mail 地址!");
              document.form1.email.focus();
              return false;
          }
         if (document.form1.pass.value.length < 6)
          {
              window.alert("密码太短,至少输入 6 位密码!");
              document.form1.pass.focus();
              return false;
          }
       if ((document.form1.email.value.indexOf('@',0) == -1)||
   (document.form1.email.value.indexOf('.',0) == -1))
         {
            window.alert("输入 E-mail 地址有误!");
            document.form1.email.focus();
            return false;
         }
       document.form1.submit();
     }
</script>
<body>
     <form method = "get" name = "form1" action = "formdeal.jsp">
       <h1 align = center > <font face = "华文新魏" > 注册信息 </font > </h1 >
       <p> 姓名: <input type = "text" name = "name" size = "20">
                 性别: <input type = "radio" value = "男" checked name = "sex">男
                        <input type = "radio" name = "sex" value = "女">女
                 职业: <select size = "1" name = "work">
                           <option>教师</option>
                           <option>公务员</option>
                           <option>军人</option>
                           <option>工人</option>
                           <option selected>职员
                           </option>
                        </select>
      </p>
      <p> 联系电话: <input type = "text" name = "tel">
             邮箱地址: <input type = "text" name = "email"><p>
      <p> 爱好:
```

```
            <input type="checkbox" name="CC" value="旅游">旅游
            <input type="checkbox" name="CC" value="音乐" checked>音乐
            <input type="checkbox" name="CC" value="游戏">游戏
            <input type="checkbox" name="CC" value="看书">看书
            <input type="checkbox" name="CC" value="保龄">保龄
            <input type="checkbox" name="CC" value="看电影">看电影
        </p>
        <p> 密码: <input type="password" name="pass" size="20">
            确认密码: <input type="password" name="confirm" size="20"></p>
        <p> 自我描述: </p>
        <p align=center><textarea rows="8" name="des" cols="60"></textarea></p>
        <p align=center>
            <input type="button" value="提交" name="B1" onclick="check()">
            <input type="button" value="重置" name="B2" onclick="check()">
        </p>
    </form>
    </body>
    </html>
```

本例运行后，网页上的输出和例3-2相同，但在本例中，当输入的信息不符合要求时，则会弹出相应的提示框，如在注册信息中没人输入姓名，单击“提交”按钮后，则弹出如图3-3所示的提示框。本例源代码存放于本书配套素材中的formtijiao.html文件中(文件路径为code\3)。



图3-3 弹出“输入姓名”提示框

3.2 JSP基本语法

从作用上看，JSP能将网页的动态部分与静态部分有效地分开。JSP页面由三类元素组成：HTML标记、Java程序片和JSP标签组成。HTML标签用于创建用户界面，Java程序片实现逻辑计算，而JSP标签用于控制JSP页面属性。HTML标签在上一节作了讲解，本节主要讲解Java程序片和JSP标签的用法。

3.2.1 Java程序片

Java程序片是用来实现逻辑计算的，是JSP中的脚本元素，它包括3个部分，即声明、表达式和脚本代码。

1. 声明

JSP中的声明用来定义一个或多个合法的变量(包括普通变量和类变量)和方法，并不输出任何的文本。在声明中定义的变量和方法将在JSP页面被载入时初始化。

JSP声明的语法格式为：

```
<%! Java声明 %>
```

其中，在<%! 和%>标记符之间的Java声明即为声明的变量、方法名称和内容。例如：

```
<%!
    int  i,j=120;
    String str="java 程序片";
    Circle a=new circle(3.0);
    Date  date;
%>
```

在声明变量和方法时，有以下几点需要注意。

（1）声明必须以“；”结尾，在程序中可以一次性声明多个变量，只要以分号结尾就可以了。

（2）一个声明仅在一个页面中有效。对于每个页面都要用到的声明，可以把它们写成一个单独的文件，然后用<%@include%>或<jsp:include>将它们包含进来。

（3）可以直接使用在<%@page%>中被包含进来的已经声明的变量和方法，而不需要对其重新进行声明。

2. 表达式

表达式是指一个在脚本语言中定义的表达式，JSP 中的表达式可以被看作一种简单的输出形式，可以将计算结果转换成一个字符串并输出。

JSP 表达式的语法格式为：

```
<% = 表达式 % >
```

其中这个表达式必须能计算出数据值。表达式的值由服务器负责计算，并将计算结果以字符串形式发送到客户端显示。例如：

```
<% = getDate() % >          //输出系统当前时间
<% = 3*5+9 % >              //输出 24
<%!
        int  a=30;
        int  b=40;
        int  c=50;
%>
<% = a+b+c  %>              //求 x=a+b+c 的值
```

在书写表达式时还要注意以下几点：

（1）<%=是一个完整的符号，<% 和=不能有空格。

（2）JSP 的表达式中没有分号，除非在加引号的字符串部分才使用分号。

（3）表达式能够使用任何 Java 语法，有时候也能作为其他 JSP 元素的属性值。

3. JSP 脚本代码

所谓脚本代码，即 JSP 中的代码部分，是一段 Java 程序的代码，它被插入到 JSP 所生成的目标 Servlet 的 Service 方法中，可包含多个 JSP 语句、变量和表达式。

JSP 脚本代码的语法格式为：

```
<% jsp 脚本代码(scriptlet)%>
```

用户可以在<%和%>标记符之间使用任何有效的程序片段，只要符合 Java 本身的标准语法即可。通常主要的程序也是写在这里面，JSP 脚本代码主要用于 3 个方面，即：

（1）声明将要用的变量。

（2）显示表达式。

（3）使用内部对象和使用<jsp:useBean>声明过的对象，编写 JSP 程序。

【例 3-4】 计算并输出表达式的值。

本程序有两个脚本代码块。变量 i 是全局变量，在整个 JSP 页面内有效，x、y、z 是局部变量，在本 JSP 页面内的所有 Java 代码块中有效。

```
<!-- 例程 3-4jisuan.jsp -->
<%!
    int i;              //定义全局变量 i
%>

<!-- 下面是第 1 个 Java 代码块 -- >
<%
    int x = 3;          //定义局部变量 x
%>

<!-- 下面是第 2 个 Java 代码块 -- >
<%
    int y = 3;          //定义局部变量 y
    int z = 4;          //定义局部变量 z
    i = x + y + z;      //计算表达式的值
    out.print(i);       //输出 i 的值
%>
```

本例源代码存放于本书配套素材中的 jisuan.jsp 文件中(文件路径为 code\3)。

【例 3-5】 计算圆的面积和周长。

在本例中，由客户提供圆的半径，然后计算圆的面积和周长。

本例程序算法为：先定义一个圆类 Circle，该类包含计算面积和周长的方法。使用表单创建用户输入圆半径的界面，通过表单获得用户输入的圆的半径 r，然后以半径 r 为参数创建一个圆对象，计算圆的面积和周长，最后输出计算结果。

```
<! — 例程 3-5 circle.jsp -- >

<%@ page contentType = "text/html;charset = GB2312" %>
<html>
 <body bgcolor = "yellow"><font size = 4 color = "blue">
  <p align = "center">
  <b>请输入圆的半径:</b>
  <br>
  <form action = ""method = get name = form>
       <input type = "text" name = "cat" value = "1">
       <input type = "submit" value = "计算" name = submit>
  </form>
  <%!
      public class Circle
```

```
        {
          double r;
          Circle(double r)
          {
            this.r = r;
          }
          double area()
          {
            return Math.PI * r * r;
          }
          double zhou()
          {
            return Math.PI * 2 * r;
          }
        }
      %>
     <%
          String str = request.getParameter("cat");
          double r;
          if(str! = null)
            {
              r = Double.parseDouble(str);
            }
          else
            {
              r = 1;
            }
            Circle circle = new Circle(r);          //创建对象
      %>
    <p align = "center"> 圆的面积为: <% = circle.area() %>
    <p align = "center"> 圆的周长为: <% = circle.zhou() %>
    </font>
    </body>
  </html>
```

本程序的输出结果如图 3-4 所示。本例源代码存放于本书配套素材中的 circle.jsp 文件中(文件路径为 code\3)。

专家点拨：如果要在脚本代码内部使用字符%>，必须写成%\>。



图 3-4　程序 circle.jsp 的输出结果

3.2.2　JSP 标签

一般使用 JSP 标签控制页面属性。JSP 标签分为 JSP 注释、JSP 指令标签和 JSP 动作标签 3 类。

1. JSP 注释

为了便于程序的阅读，需要在编写代码时书写注释，注释本身不产生语句功能，只用来

增强 JSP 文件的可读性，便于用户维护 JSP 文件。JSP 注释分两种：一种是在客户端显示的注释，称为 HTML 注释；另外一种就是客户端看不到，只给开发人员专用的注释，称为 JSP 注释，也称为隐藏注释。

1）HTML 注释

JSP 页面使用这种注释时，客户端通过浏览器查看 JSP 源文件时，能够看到 HTML 注释文字。其语法格式是：

```
<!-- 注释 -- >
```

2）JSP 注释

这是 JSP 的标准注释，写在 JSP 程序中，使用这种注释时，JSP 引擎编译该页面时会忽略 JSP 注释，在返回的 HTML 源代码中也看不到，这种注释主要供编程人员使用。其语法格式为：

```
<% -- 注释 -- % >
```

【例 3-6】 比较 HTML 注释和 JSP 注释。在本例中分别运用了 HTML 注释和 JSP 注释，注意它们运行后的不同效果。

```
<!-- 例程 3-6 zhushi.jsp -- >
<%@ page contentType = "text/html; charset = gb2312" %>
<h3>HTML 注释测试</h3>
<!-- This ip <% = request.getRemoteAddr() %> -->

<h3>JSP 注释测试</h3>
<% -- 此注释不会在源代码中显示<% = request.getRemoteAddr() %> -- %>
```

本例程运行后返回的 HTML 源代码如图 3-5 所示。本例源代码存放于本书配套素材中的 zhushi.jsp 文件中（文件路径为 code\3）。

从图 3-5 中可以看到，HTML 注释在返回的 HTML 源代码中能看到，HTML 注释中的 JSP 表达式可以被执行。而 JSP 注释则不能从源文件中看到。这主要在于 JSP 容器不会对<%--和 --%>之间的语句进行编译，它不会显示在客户端的浏览器上。

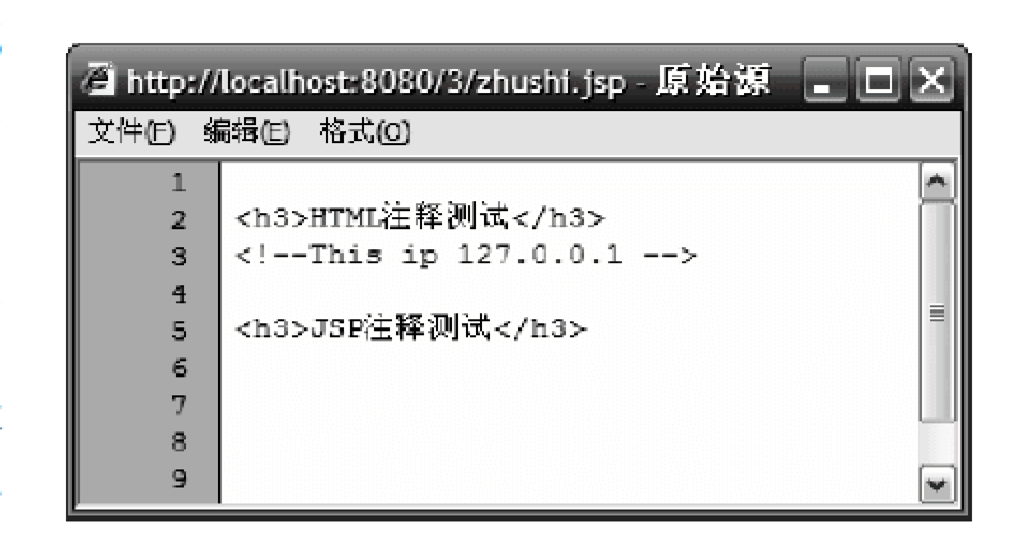


图 3-5　例程 zhushi.jsp 返回 HTML 的源代码

2. JSP 指令标签

JSP 指令标签是为 JSP 引擎设计的，它并不向客户端产生任何输出，所有的指令只在当前的 JSP 页面中有效。主要用来提供整个 JSP 网页相关的信息，并且用来设定 JSP 网页的相关属性，如网页的编码方式、是否为错误处理页面等信息。

JSP 指令标签主要包括 page 和 include 指令。

1）page 指令标签

page 指令主要用来定义整个 JSP 页面的各种属性，它描述了与页面相关的一些信息，

作用域为它所在的 JSP 文件页面和其包含的文件。

一个 JSP 页面可以包含多个 page 指令，在指令中，除了 import 属性外，每个属性只能定义一次，否则 JSP 页面编译将出现错误。

page 指令标签由多个属性名="属性值"对构成，通过这种方式设置页面的属性，其语法格式如下：

```
<%@ page
[language = "java"]
[extends = "classname"]
[import = "packname/classname"]
[session = "true/false"]
[buffer = "none/sizekb"]
[autoFlush = "true/false"]
[isThreadeSafe = "true/false"]
[info = "info_text"]
[errorPage = "error_url"]
[isErrorPage = "true/false"]
[contentType = "MIME_type"]
[pageEncoding = "   "]
%>
```

【说明】

(1) language="java"：language 属性定义了 JSP 页面中所使用的脚本语言。目前 JSP 必须使用的是 Java 语言，因此该属性的默认值为 java，因此也要求 JSP 页面的编程语言必须符合 Java 语言规则。language 属性设置为 language="java"。

专家点拨：使用该属性需要注意的是，在第一次出现脚本元素之前，必须设置该属性的参数值，否则将会导致严重的错误。

(2) extends="classname"：该属性定义 JSP 页面产生的 Servlet 所继承的父类。由于该属性将限制 JSP 引擎提供特定的超类，这些超类可能会改善所提供服务的品质，因此在使用该属性时必须慎重。

(3) import="packname/classname"：该属性是所有 page 属性中唯一可以多次设置的属性，而且累加每个设置，它描述了 JSP 脚本环境中要使用的类。

(4) session="true/false"：该属性指定 JSP 页面是否参与一个 http 会话，它的默认值是 true，表示该属性所在的页面参与指定 http 的会话，为 false 时，则不参与。

(5) buffer="none/sizekb"：该属性指定 JSP 网页的缓冲区大小，默认的缓冲区大小为 8KB。如果该属性值为 none，将不缓冲，所有的响应输出都通过 PrintWriter 直接写到 ServletResponse 中。

(6) autoFlush="true/false"：该属性的默认值为 true，表示当缓冲区满时，到客户端的输出将会自动刷新，若该属性为 false，则当缓冲区满时，将会出现缓冲区溢出异常。

专家点拨：在 buffer 属性值取 none 的时候，autoflush 属性值不能设置为 false。因为当 buffer 取值为 none 时，表明没有设置缓冲区，因而 JSP 页面的 jspWriter 本身将会自动刷新缓冲区。

(7) isThreadeSafe="true/false"：该属性用来设置 JSP 页面是否可以多线程使用，其

默认值为 true,如果采用默认值,则在运行 JSP 页面时,可能会同时接受多个客户的请求。当该属性取值为 false 时,一个 JSP 处理器将会逐个地接受客户的请求。

(8) info="info_text":在该属性中定义了一个任意的字符串,可以用来说明 JSP 页面中待说明的信息,该字符串将会直接加入到翻译好的页面中,可以通过 Servlet.getservletInfo()方法获得该属性的值。

(9) errorPage="error_url":该属性用于表示当发生异常错误时调用的 JSP 页面,当页面出现一个没有被捕获的异常时,错误信息将以 throw 语句抛出,而被设置为错误信息网页的 JSP 页面,将利用 exception 隐含对象,取得错误信息,通常默认忽略。

(10) isErrorPage="true/false":该属性定义了当前的 JSP 页面是否为另外一个 JSP 页面错误显示的目标,默认值为 false。当设置为 true 时,JSP 页面将可存取隐含的 exception 对象,并通过该对象取得从发生错误的网页传出的错误信息。

(11) contentType="MIME_type":该属性定义了 JSP 页面及其相应的字符编码以及 JSP 页面响应的 MIME 类型。TYPE 的默认值为 text/html,字符编码的默认值为 ISO-8859-1。

(12) pageEncoding="　":该属性描述 JSP 页面的字符编码,通常默认值为 ISO-8859-1。

(13) isELIgnored 属性:该属性用来设置表达语言(Expression Language,EL,已纳入 JSP 2.0,或为标准规范之一)是否被忽略,若取值为 true,则忽略 EL 表达式计算,反之则不忽略。

【例 3-7】 利用 page 指令显示当前系统时间。

本例使用 page 指令的 import 属性指定了其脚本环境中可以使用的类,以便在程序中可以调用 Java 中的日期函数 date()。

```
<!-- 例程 3-7 datetime.jsp -->

<%@ page contentType="text/html; charset = gb2312" language="java" %>
<%@ page import="java.util.*,java.lang.*" %>      //指定脚本环境中的类
<%@ page errorPage="" %>
<html>
  <head>
     <title>page 测试</title>
  </head>
  <body bgcolor = "white">
     <font size=4 color = "blue">
        现在的时间是: <% =(new java.util.Date()).toLocaleString() %>
 //使用 date()函数显示当前系统日期
    </font>
  </body>
</html>
```

该程序运行效果如图 3-6 所示。本例源代码存放于本书配套素材中的 datetime.jsp 文件中(文件路径为 code\3)。

现在的时间是: 2009-9-14 23:39:41

图 3-6　datetime.jsp 的运行效果

2) include 指令标签

在 JSP 中用 include 指令在标签位置处静态插入一个

文件,同时解析这个文件中的 JSP 语句。所谓静态插入指用被插入的文件内容代替该指令标签与当前 JSP 文件合并成新的 JSP 页面后,再由 JSP 引擎转译为 Java 文件。

使用 JSP 的 include 指令有助于实现 JSP 页面的模块化,该指令的语法格式如下:

```
<%@  include file = "filename" %>
```

其中,filename 是指被插入的文件名称,这个文件可以是 JSP 网页、HTML 网页、文本文件,或是一段 Java 程序。这个被插入的文件的路径一般来说是指相对路径,不需要什么端口、协议和域名,如 cons.jsp、/beans/calend.jsp 等。如果路径是以文件名或目录名开头,那么这个路径就是正在使用的 JSP 文件的当前路径;如果路径以"/"开头,则该路径主要是参照 JSP 应用的上下关系路径。

被插入的文件要求满足以下条件:

(1) 被插入的文件必须与当前 JSP 页面在同一个 Web 服务目录下。

(2) 被插入的文件与当前 JSP 页面合并后的 JSP 页面必须符合 JSP 语法规则。

【例 3-8】 在 sqrt.jsp 页面中静态插入一个 qiuzhi.jsp 文件。

本例调用 include 指令标签插入一个求平方根的文件 qiuzhi.jsp,在主文件 sqrt.jsp 中只显示一行要求输入数字的文字,而求值运算则在被插入的文件中,相当于程序的函数调用。

```
<!-- 例程 3 - 8 sqrt.jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<HTML>
 <BODY Bgcolor = "cyan">
  <FONT size = 4 >
    <CENTER>
        <p>请在下方的文本框输入一个正数,单击"计算"按钮求该数的平方根:
        <%@ include file = "qiuzhi.jsp" %>    //插入文件 qiuzhi.jsp
    </CENTER>
  </FONT>
 </BODY>
</HTML>

<!-- 被插入的静态文件 qiuzhi.jsp -->
<%@ page contentType = "text/html;charset = GB2312" %>
    <FORM action = ""   method = post   name = from>
       <INPUT type = "text"    name = "ok" > <BR>
       <INPUT TYPE = "submit"  value = "计算"  name = submit>
    </FORM>
    <%
       String a = request.getParameter("ok");
       if(a == null)
         {
a = "1";
         }
       try
         {
            double number = Integer.parseInt(a);
            out.print("<BR>" + Math.sqrt(number));
```

```
        }
    catch(NumberFormatException e)
        {
          out.print("<BR>" + "请输入数字字符");
        }
%>
```

程序运行效果如图 3-7 所示。本例源代码存放于本书配套素材中的 sqrt.jsp 文件及 qiuzhi.jsp 文件中(文件路径为 code\3)。

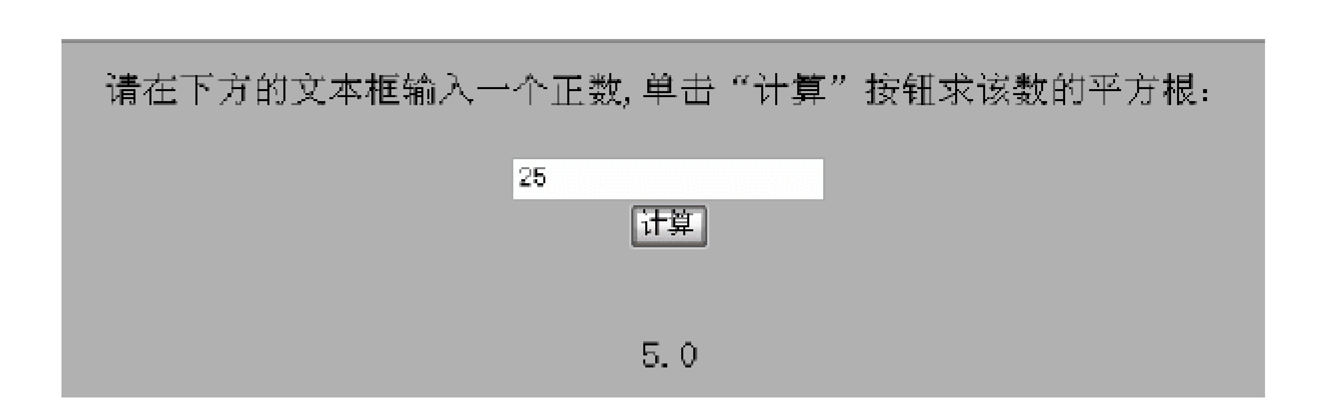


图 3-7　sqrt.jsp 的运行效果

【例 3-9】 用 include 指令标签显示当前 IP 地址。

本例也是使用 include 指令标签插入一个显示当前 IP 地址的文件。

```
                    <!-- 例程 3-9 include.jsp -- >
<%@ page contentType = "text/html; charset = gb2312" %>
<body bgcolor = "white">
  <font color = "blue">
    您的 IP 是:
    <%@include file = "ip.jsp" %>            //插入文件 ip.jsp
  </font>
</body>
                    <!-- 被插入的例程文件 ip.jsp -->
<% = request.getRemoteAddr() %>
```

您的IP是：127.0.0.1

图 3-8　include.jsp 的运行结果

该程序的运行结果如图 3-8 所示。本例源代码存放于本书配套素材中的 include.jsp 文件及 ip.jsp 文件中(文件路径为 code\3)。

从以上两个例题可以看出,由于使用了 include 指令标签,可以把一个复杂的页面分成若干个简单的部分,这样大大增加了 JSP 页面的可管性,当要对页面更改时,只需要更改对应的部分就可以了。

一个网站的导航栏往往是一致的,所以可以将导航栏写成一个单独的页面,然后通过指令实现对导航栏的导入。

include 指令是在 JSP 页面被编译时导入文件的。所以当被包含文件变化的时候,需要重新编译包含的 JSP 页面。因此,如果要经常改变导航栏的 Web 程序,则使用 include 指令标签就不方便了。这时可以使用<jsp:include>来完成工作。

3.2.3 JSP 的动作指令

JSP 的动作和 JSP 的指令标签不同,它是客户端请求时动态执行的,是通过 XML 语法格式的标记来实现控制 Servlet 引擎行为的。利用 JSP 的动作可以实现很多功能,包括动态

地插入文件、重用 JavaBean 组件、把用户重定向到另外的页面、为 Java 插件生成 HTML 代码等。

1. <jsp:include>动作指令

<jsp:include>动作用来把指定文件插入正在生成的页面中，其语法如下：

```
<jsp:include page="被包含的文件 relativeURL"  flush="true|false">
  <jsp:param name="parameterName" value="parameterValue">
</jsp:include>
```

【说明】

(1) page 参数指定被包含文件的路径，可以是文档相对路径或站点相对路径，也可以是表示相对路径的表达式。

(2) flush 属性指定在读入包含内容之前是否清除当前缓冲区。默认值为 false。

(3) <jsp:param>动作设置要传递到被包含的动态文件的参数，name 和 value 分别表示参数的名称和值。

若不向 JSP 页中传递参数，则 include 动作可以简化为以下形式：

```
<jsp:include page="relativeURL"  flush="true"|"false" />
```

上一节介绍的 include 指令标签，它是在 JSP 文件被转换成 Servlet 的时候引入文件的，而这里的<jsp:include>动作不同，它是在页面被请求的时候插入文件的。

<jsp:include>动作允许包含静态文件和动态文件，若被包含文件是静态的 HTML 文件，则执行该动作的结果仅仅将被包含文件的内容添加到 JSP 文件中，被包含文件不会被 JSP 引擎执行。若被包含文件也是动态的 JSP 文件，则这个被包含文件将被 JSP 引擎执行，而且可以利用<jsp:param>动作向被包含文件传递参数。由于<jsp:include>动作能够同时静态和动态这两种文件，因此在需要使用这个动作指令插入文件时，要先判断该文件是动态的还是静态的。

<jsp:include>动作的文件引入时间决定了它的效率要稍微差一点，而且被引用的文件不能包含某些 JSP 代码(例如不能设置 HTTP 头)，但它的灵活性却要好得多。

【例 3-10】 用<jsp:include>动作指令显示当前 IP 地址。

```
                         <!-- 例程 3-10 jspinclude.jsp -->
<html>
  <head>
    <title>include 测试</title>
  </head>
<%@ page contentType= "text/html; charset = gb2312" %>
  <body bgcolor = "cyan">
    <font color = "red">
         IP 地址是:
           <jsp:include page = "ip.jsp" />      //动态导入 IP 地址文件 ip.jsp
         <br>
       静态的 IP 是:
           <jsp:include page = "ip.html" />     //插入静态文件 ip.html
    </font>
```

```
  </body>
</html>
```

```
<!-- 例程 3 - 11 ip.html -->
172.168.102.42
```

IP地址是：127.0.0.1
静态的IP是：172.168.102.42

图 3-9　include 动作测试效果

程序的运行效果如图 3-9 所示。本例源代码存放于本书配套素材中的 jspinclude.jsp 文件、ip.jsp 文件及 ip.html 文件中(文件路径为 code\3)。

从此例的程序可以看出，<jsp:include>动作是程序执行时动态导入的，当被包含的文件改变时，程序运行也会发生相应的变化。

2. <jsp:forward>动作指令

<jsp:forward>动作指令用于将客户端请求从当前 JSP 页重定向到另一个页面，语法如下：

```
<jsp:forward page="要重定向的页面 relativeURL">
   <jsp:param name="parameterName" value="parameterValue" />
</jsp:forward>
```

【说明】

(1) page 属性是指定要重定向的目标页面的 URL，目标页面可以是 HTML 静态页、JSP 动态页、Servlet 或其他可处理 request 对象的文件，路径可以采用文档相对路径或站点相对路径，也可以是表示相对路径的表达式。

(2) <jsp:param>动作用于设置要传递到动态文件的参数，若要向动态文件传递多个参数，可以添加<jsp:param>动作。

若不向目标页中传递参数，则 forward 动作可以简化为以下形式：

```
<jsp:forward page="relativeURL" />
```

【例 3-11】　forward 动作测试。

在本例中，文件 forward.jsp 中有三句文本，分别是"文件将定向到 newworld.jsp"、"这两句话被执行了，但是看不到，因为页面重定向了"和"重定向语句后面的内容将不被执行"，但由于在程序中使用了<jsp:forward page = "newworld.jsp" />，使得文件直接被重定向到文件 newworld.jsp，转而执行文件 newworld.jsp 的内容。

```
<!-- 例程 3 - 12 forward.jsp -->
<html>
  <head>
    <title>forward 测试</title>
  </head>
  <%@ page contentType= "text/html; charset = gb2312"%>
  <body>
       文件将定向到 newworld.jsp
       这两句话被执行了,但是看不到,因为页面重定向了
       <jsp:forward page = "newworld.jsp" />     // 调用<jsp:forward>进行重定向
       重定向语句后面的内容将不被执行
</body>
```

```
</html>

                    <!-- 例程 3-13 newworld.jsp -->
<html>
  <head>
    <title>forward 测试</title>
  </head>
  <%@ page contentType= "text/html; charset = gb2312" %>
  <body>
      欢迎来到这儿,这是一个全新的世界!!!
  </body>
</html>
```

本程序执行效果如图 3-10 所示。本例源代码存放于本书配套素材中的 forward.jsp 文件及 newworld.jsp 文件中(文件路径为 code\3)。



图 3-10 forward 测试效果

从此例中可以看到,<jsp:forward>动作指令的作用是,当前页面执行到该指令处后转向其他 JSP 页面执行,<jsp:forward>动作以下的代码不被执行。

3. <jsp:plugin>动作指令

<jsp:plugin>动作指令指示 JSP 页面加载 Java plugin 插件,该插件由客户负责下载,并使用该插件执行一个 applet(Java 小应用程序)或 JavaBean,其语法格式如下:

```
<jsp:plugin type= "bean" | "applet" code= "classFileName"
   codebase= "classFileDirectoryName"
   [ name= "instanceName" ] [ archive= "URIToArchive, ..." ]
   [ align= "bottom" | "top" | "middle" | "left" | "right" ]
   [ height= "displayPixels" ] [ width= "displayPixels" ]
   [ hspace= "leftRightPixels" ] [ vspace= "topBottomPixels" ]
   [ jreversion= "JREVersionNumber" ]
   [ nspluginurl= "URLToPlugin" ] [ iepluginurl= "URLToPlugin" ] >
  [ <jsp:params>
        <jsp:param name= "parameterName" value= "parameterValue" />
     </jsp:params> ]
    [ <jsp:fallback> text message for user </jsp:fallback> ]
</jsp:plugin>
```

【说明】

(1) type="bean"|"applet":设置将被执行插件对象的类型,必须指定这个属性的值是bean还是applet,这个属性没有默认值。

(2) code="classFileName":指将被plugin执行的Java类的名字,必须以.class结尾,并且这个文件必须存放在codebase属性所指定的目录中。

(3) codebase="classFileDirectoryName":指定将被执行的Java类的目录,默认值为使用<jsp:plugin>的JSP网页所在的目录。

(4) name="instanceName":指定bean或applet实例的名字,用于在JSP页面的其他部分调用时使用,这使得被同一个JSP文件调用的bean或applet之间的通信成为可能。

(5) archive="URIToArchive,...:这是一些用逗号分开的路径名,用于预装一些将要使用的类,这样可以提高applet的性能。

(6) align="bottom"|"top"|"middle"|"left"|"right":指明图形、对象和applet在浏览器中的位置。

(7) height="displayPixels",width="displayPixels":设定bean或applet将要显示的长、宽的值,此值为数字,单位为像素。

(8) hspace="leftRightPixels" ,vspace="topBottomPixels":设定bean或applet显示时在屏幕左右上下所需留下的空间,单位为像素。

(9) jreversion="JREVersionNumber":指明bean或applet运行时所需的Java运行环境,即JRE的版本。默认值是1.1版本。

(10) nspluginurl="URLToPlugin":这是Netscape Navigator用户能够使用的JRE的下载地址,此值为一个标准的URL。

(11) iepluginurl="URLToPlugin":这是IE用户能够使用的JRE的下载地址。

(12) <jsp:params>:指定需要向bean或applet传送的参数或参数值,由name指定参数名,value指定参数值。

(13) <jsp:fallback>:这是用于在Java插件不能启动时显示给用户的信息,在运行过程中若插件不能启动,则浏览器会显示<jsp:fallback>标签中的信息,以提示出错。

【例3-12】 用plugin插件运行Applet程序。

本例在plugin.jsp页面中调用applet1.java小程序以实现算术运算。本例源代码存放于本书配套素材中的plugin.jsp文件及applet1.java文件中(文件路径为code\3)。

```
<!-- 例程 3-14 plugin.jsp -->
<%@ page contentType = "text/html; charset = gb2312" %>
<HTML>
  <HEAD>
    <TITLE>用 plugin 加载 Applet 的例子</TITLE>
  </HEAD>
  <BODY>
    <center>
    <FONT SIZE = 4 COLOR = red>用 plugin 加载 Applet 的例子 </FONT>
    <BR>
    <!-- 用 plugin 加载 applet -->
    <jsp:plugin type = "applet" code = "Applet1.class" height = "60" width = "420"/>
```

```
    </BODY>
</HTML>
                          <!-- 例程 3-15 applet1.java -->
import java.applet.*;
import java.awt.*;
public class Applet1 extends Applet
{
    Label l1;
    public void init()
    {
      int x = 9;
      int y = 20;
      int z = x + y;
      l1 = new Label(x + y + "的和为: " + z);
     add(l1);
   }
}
```

3.3 上机指导与练习

3.3.1 计算三角形面积并对程序进行注释

1. 练习目标

(1) 熟悉用 HTML 语言来创建表单。

(2) 熟悉 HTML 注释及 JSP 注释的用法以及二者的区别。

2. 练习指导

在该练习程序中，通过表单要求用户输入三角形 3 条边的长度，并判断这 3 条边能否构成一个三角形，若能构成三角形，就根据三角形面积计算公式计算面积，最后输出计算结果。

(1) 创建表单。表单中包含 3 个文本框，从 3 个文本框获取 3 条边的长度。

(2) 判断 3 条边能否构成一个三角形，如果不能构成三角形，则输出信息“不能构成三角形”。

(3) 如果能构成一个三角形，则计算三角形的面积并输出。

该程序的源代码如下：

```
                      <!-- 例程 3-16sanjiaoxing.jsp -->
<%@ page contentType = "text/html;charset = GB2312" %>
<HTML>
<BODY>
<font face = "楷体_GB2312",size = "8" color = "blue">
<P>请输入三角形的 3 条边 a、b、c 的长度:
<BR>
<!-- 以下是 HTML 表单,向服务器发送三角形的 3 条边的长度(本条语句是 HTML 注释) -->
  <FORM action = "" method = post   name = form>
```

```
        <P>请输入三角形边 a 的长度:
            <INPUT type="text" name="a">
            <BR>
        <P>请输入三角形边 b 的长度:
            <INPUT type="text" name="b">
            <BR>
        <P>请输入三角形边 c 的长度:
            <INPUT type="text" name="c">
            <BR>
        <p><center><INPUT TYPE="submit" value="计算" name=submit></center>
    </FORM>

<%-- 获取客户提交的数据(本条语句是 JSP 注释)--%>
<%
    String string_a=request.getParameter("a");
    String string_b=request.getParameter("b");
    String string_c=request.getParameter("c");
    double a=0,b=0,c=0;
%>

<%-- 判断字符串是否是空对象,如果是则初始化(本条语句是 JSP 注释)--%>
<%
  if(string_a==null)
  {
      string_a="0";
      string_b="0";
      string_c="0";
  }
%>

<%-- 求出边长,并计算面积(本条语句是 JSP 注释)--%>
<%
   try {                                         // 用于捕获异常
       a=Double.valueOf(string_a).doubleValue();
       b=Double.valueOf(string_b).doubleValue();
       c=Double.valueOf(string_c).doubleValue();
        if(a+b>c&&a+c>b&&b+c>a)
         {
         double p=(a+b+c)/2.0;
         double mianji=Math.sqrt(p*(p-a)*(p-b)*(p-c));
           out.print("<BR>"+"三角形面积:"+mianji);
         }
        else
         {
         out.print("<BR>"+"您输入的三条边不能构成一个三角形");
         }
       }
   catch(NumberFormatException e)
     {
           out.print("<BR>"+"请输入数字字符");
       }
```

```
  %>
</font>
</BODY>
</HTML>
```

本例源代码存放于本书配套素材中的 sanjiaoxing.jsp 文件中(文件路径为 code\3\上机指导)。

3.3.2 求 1 到 100 的连续和

1. 练习目标

(1) 熟悉动态加载页面的运用。
(2) 熟悉 include 动作指令的使用。
(3) 熟悉参数传递的使用。

2. 练习指导

本程序由两个页面构成,由主页面动态加载次页面,并传递参数 100 给次页面,由次页面实现求和功能。

(1) 主页面 qiuhe.jsp 动态加载页面 addnum.jsp。
(2) 给参数 Computer 传递数据 100。
(3) 次页面 addnum.jsp 获得参数 Computer 的值。
(4) 求 1 到 Computer 的和。
(5) 输出数据。

该程序的源代码如下:

```
                    <!-- 例程 3-17 qiuhe.jsp -->
<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY  BGCOLOR=Cyan>
    <jsp:include page="addnum.jsp">
    <jsp:param name="computer" value="100"/>
    </jsp:include>
</BODY>
</HTML>
                    <!-- 例程 3-18 addnum.jsp -->
<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY>
  <%
      String str=request.getParameter("computer"); //获取值
      int n=Integer.parseInt(str);
      int sum=0;
      for(int i=1;i<=n;i++)
         {
           sum=sum+i;
         }
```

```
 %>
 <P>从 1 到<% = n %>的连续和是:<% = sum %>
</BODY>
</HTML>
```

本程序源代码存放于本书配套素材中的 qiuhe.jsp 文件及 addnum.jsp 文件中(文件路径为 code\3\上机指导)。

3.3.3 输出 0~1 之间的任意随机数

1. 练习目标

(1) 熟悉 forward 指令的使用。

(2) 熟悉重定向页面。

2. 练习指导

本程序由两个页面构成,由一个主页面 suiji.jsp 产生随机数并传递给另一重定向页面,重定向页面 redi.jsp 获得随机数并输出此数据。

(1) 在主页面 suiji.jsp 中利用 Math.random()产生随机数 n。

(2) 使用<jsp:forward>进行重定向,转向页面 redi.jsp。

(3) 使用<jsp:param>进行参数传递,将参数 n 的值传递给 redi.jsp 页面。

(4) 重定向页面 redi.jsp 获得 n 的值,并输出 n 的值。

该程序的源代码如下:

```
<!-- 例程 3-19suiji.jsp -->
<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY>
 <%
       double i=Math.random();
 %>
     <jsp:forward page="redi.jsp">
       <jsp:param  name="n" value="<%=i%>" />
     </jsp:forward>
</BODY>
</HTML>
```

```
<!-- 例程 3-20 redi.jsp -->
<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY bgcolor=cyan>
     <FONT size=3>
     <%
        String str=request.getParameter("n");
        if(str==null) str="0";
        double n=Double.parseDouble(str);
     %>
        <P>您传过来的数值是:<%=n%>
     </FONT>
```

```
</BODY>
</HTML>
```

本程序源代码存放于本书配套素材中的 suiji.jsp 文件及 redi.jsp 文件中(文件路径为 code\3\上机指导)。

本章小结

本章对使用 JSP 编程所需的基础知识 HTML 和 JavaScript 进行了简要的介绍。HTML 主要用于创建用户界面,而 JavaScript 则是一种基于对象和事件驱动的脚本语言。另外,本章还介绍了 JSP 的基本语法和 JSP 的指令标签和动作标签。通过对本章的学习,读者可以为以后的学习打好基础。

习题 3

一、简答题

1. 简述 HTML 标签的作用。
2. JSP 指令标签有哪些? 分别起什么作用?
3. 在 JSP 文件中 HTML 注释和隐藏注释有什么区别?

二、操作编程题

1. 编写 HTML 文件,要求输出如下的学生信息表。

学号	姓名	性别	出生日期	籍贯	系部	专业	班级	联系电话
1001	张青	男	1985.10.09	南京	计算机	电子商务	06 电子商务(1)班	13502291568
1002	赵阳	男	1986.03.25	武汉	外语	商务英语	06 商务英语(2)班	13800662598

2. 改写例 3-2 的表单 form.html,加入一个年龄的文本框,并使用 JavaScript 验证其输入是 16 至 100 之间的数字。

3. 编写一个程序,客户在窗口中输入圆柱体的底半径和高度,提交数据后,计算出圆柱体的表面积和体积。

第4章 JSP内置对象

JSP 中内置了一些对象，内置对象也称为内部对象或隐藏对象，使用 JSP 内置对象，可以方便地操作页面，访问页面环境，实现页面内、页面间、页面与环境之间的通信。本章将分别介绍这些内置对象的使用方法。

本章主要内容：

- 内置对象的概念；
- 内置对象的作用域和生命期；
- 内置对象的作用和关系；
- 内置对象的使用方法。

4.1 JSP 内置对象概述

为了方便 Web 程序的开发，在 JSP 页面中内置了一些默认的对象，这些对象不需要预先声明和创建实例就可以在脚本代码和表达式中使用。

4.1.1 JSP 内置对象的来源

有些成员变量不用声明就可以在 JSP 页面的脚本（如 Java 程序片和 Java 表达式）中使用，这就是所谓的内置对象。

使用 JSP 动态网页开发实现 Java Web 应用时，JSP 文件通过 JSP 引擎翻译为 Servlet 文件。Servlet 是一个 Java 代码文件，用于定义一个继承 HttpJspBase 类的子类，并创建各个 JSP 内置对象。因此，在设计 JSP 动态网页时可以直接使用这些对象。

4.1.2 JSP 内置对象介绍

JSP 的内置对象是由 JSP 容器自动生成的，在 JSP 页中可以直接使用而无须进行声明。在 JSP 动态网页设计过程中，灵活地应用这些内置对象，可以实现许多实用的功能。

在不需要显示声明的情况下，每一个 JSP 页面中可以使用的内置对象有 9 个，即 request、response、session、out、application、config、pageContext、page 以及 exception。

4.2 request 对象

request 对象主要用于接收客户端通过 HTTP 协议连接传输到服务器端的数据。当客户访问服务器页面时，会提交一个 HTTP 请求，request 对象就是对这个 HTTP 请求包的封装，该请求中包含所有客户端传送给服务器端的数据，如请求的来源、Cookies 以及请求的相关参数值等，其作用域就是一次 request 请求。在客户端的请求中如果有参数，则该对象就有一个参数列表。

4.2.1 HTTP 请求包

一般说来，一个 HTTP 请求包括 3 个部分：一个请求行、多个请求头和信息体。

1. 请求行

规定了请求的方法(如 get、post、head、delete、put 等)，请求的资源和使用的 HTTP 协议版本号。

2. 请求头

请求头主要说明请求客户的主机(IP)、信息体和附加信息。一个 HTTP 请求可以包括多个请求头。

3. 信息体

指请求的正文。如表单数据被封装为信息体。

下面是一个简单的 HTTP 请求包的组成：

```
Get/hello.htm HTTP/1.1                 : 请求行
Host:www.163.com                       : 请求头
Name 张平(数据组件接受的信息)           : 信息体(表单中的数据信息)
```

4.2.2 request 对象的常用方法

前面提到 request 对象就是对 HTTP 请求包的封装，因此，使用 request 对象的方法，可以获取客户端和服务器端的信息。如客户端主机名、IP 地址、传递参数名、参数值、服务器主机名和 IP 地址等。

request 对象包括很多方法，主要有以下几种。

(1) getProtocol()：用于获取客户向服务器提交信息请求时所使用的通信协议，如 http/1.1 等。

(2) getServletPath()：用于获取客户请求的 JSP 页面文件所在的目录。

(3) getContentLength()：用于获取客户请求的整个信息的长度，以字节为单位。如果无法得到该请求的长度，则返回－1。

(4) getMethod()：用于获取表单提交信息的方式，一般方式有 POST、GET、PUT 等

类型。

(5) getHeader(String s):用于获取 HTTP 头文件中由参数 s 指定的头名字的值。一般来说,s 参数可取的头名有 accept、referrer、accept-language、content-type、accept-encoding、user-agent、host、cookie 等,比如,s 取值 user-agent,将获得用户的浏览器的版本号等信息。

(6) getHeaderNames():用于获取所有 request Header 的名字,结果集是一个枚举类的实例。

(7) getHeaders(String s):用于获取头文件中指定头名字的全部值的一个枚举。

(8) getRemoteAddr():用于获取客户端的 IP 地址。

(9) getRemoteHost():用于获取客户端主机的名称,如果获取不到,就获取 IP 地址。

(10) getServerName():用于获取服务器的名称,如果没有设定服务器名,则获取服务器的 IP 地址。

(11) getServerPort():用于获取服务器的端口号。

(12) getParameter(String name):用于获取客户端传递给服务器端由 name 指定的参数值。当传递给此方法的参数名没有实际参数与之对应时,则返回 null。

(13) getParameterNames():用于获取客户提交的信息体部分中 name 参数值的一个枚举。

(14) getParameterValues(String name):用于获取由 name 指定的参数的所有值。

(15) getCookies():返回客户端的 Cookies 对象,结果是一个 Cookie 数组。

(16) getRequestURL():获得发出请求字符串的客户端地址。

4.2.3 request 对象应用实例

客户通常使用 HTML 表单向服务器的某个 JSP 页面提交信息,一般格式为:

```
<form  method = get/post  action = "接受新的页面文件">
    [接受数据的组件(0~n 个)]
    [数据提交控件]
</form>
```

【例 4-1】 获取客户提交的信息。

在主页面 input.jsp 中创建表单,提供信息输入窗口,包含一个文本控件和一个提交控件。当用户提交信息后,页面定向到另一个页面 outinfo.jsp 中,通过使用 request 对象的 getParameter()方法获取客户端的表单信息,并输出文本框和提交控件的值。

本程序的源代码如下:

```
                    <!-- 例程 4-1 input.jsp -->
<%@ page contentType = "text/html;charset = GB2312" %>
<html>
 <body bgcolor = cyan>
  <font size = 3>
   <form action = "outInfo.jsp" method = post name = form>
    <input type = "text" name = "boy">
    <input TYPE = "submit" value = "提交" name = "submit">
   </form>
```

```
    </font>
  </body>
</html>
```

<!-- 例程 4-2 outinfo.jsp -->

```
<%@ page contentType="text/html;charset=GB2312" %>
<html>
 <body bgcolor=cyan>
  <font size=4>
   <!-- 获取客户端的表单信息 -->
   <%
       String textContent=request.getParameter("boy");     // 获取文本框的信息
       byte c[]=textContent.getBytes("ISO-8859-1");
       textContent=new String(c);
       String buttonName=request.getParameter("submit");   //获取按钮信息
       byte b[]=buttonName.getBytes("ISO-8859-1");
       buttonName=new String(b);
   %>
  <!-- 将文本信息和按钮信息输出到客户端 -->
   <P><B>获取文本框提交的信息:</B><BR>
    <%=textContent%>
   <P><B>获取按钮的名字:</B><BR>
    <%=buttonName %>
   </font>
  </body>
</html>
```

【说明】

Tomcat 的默认编码是 ISO-8859-1，在 Java 里面 char 和 string 是有编码的，例如 gb2312，而 byte 则没有。如果编码、解码方式不一样，就会出现乱码。

本例在 outinfo.jsp 文件中出现的两条语句如下：

```
byte c[]=textContent.getBytes("ISO-8859-1");
textContent=new String(c);
```

是指对 c[]内的信息用 ISO-8859-1 编码，而 textContent 返回编码后的字符串。后面的两句作用相同。

程序运行后的结果如图 4-1 所示。

当用户输入信息并单击“提交”按钮后，页面定向到 outinfo.jsp 页面，显示结果如图 4-2 所示。本例源代码存放于本书配套素材中的 input.jsp 文件和 outinfo.jsp 文件中(文件路径为 code\4)。



图 4-1　客户提交信息页面

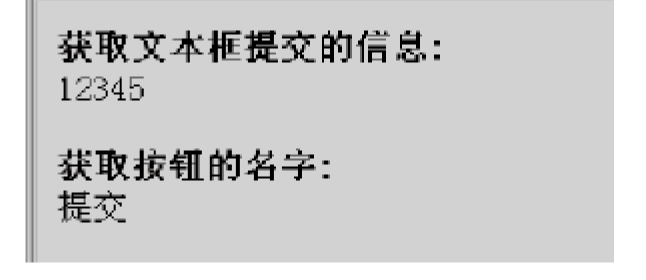


图 4-2　输出客户信息页面

【例 4-2】 获取表单提交的数据——网上单选问答题。

本例由一个页面 danxuan.jsp 输出单选试题，其表单界面中包含两组单选按钮。由信息接受页面 defen.jsp 来获取用户提交的选择，并将其与正确答案进行比较，统计得分，答对一题得一分，最后输出考试结果。

本例程序源代码如下：

```
                    <!-- 例程 4-3 danxuan.jsp -->
<HTML>
 <%@ page contentType = "text/html;charset = GB2312" %>
  <BODY>
   <FONT size = 3>
     <FORM action = "defen.jsp" method = post name = form>
       <P>1、下列作品中属于编年体历史著作的是：<BR>
       <P>
             <INPUT type = "radio" name = "r" value = "a">《国语》
             <INPUT type = "radio" name = "r" value = "b">《战国策》
             <INPUT type = "radio" name = "r" value = "c">《史记》
             <INPUT type = "radio" name = "r" value = "d">《左传》
    <P>2、具有"含泪的微笑"风格的小说家是：<BR>
    <P>
             <INPUT type = "radio" name = "P" value = "a">莫泊桑
             <INPUT type = "radio" name = "P" value = "b">契诃夫
             <INPUT type = "radio" name = "P" value = "c">欧·亨利
             <INPUT type = "radio" name = "P" value = "d">屠格涅夫
    <P>3、唐代"新乐府运动"的倡导者是：<BR>
    <P>
             <INPUT type = "radio" name = "q" value = "a">白居易
             <INPUT type = "radio" name = "q" value = "b">岑参
             <INPUT type = "radio" name = "q" value = "c">韩愈
             <INPUT type = "radio" name = "q" value = "d" >柳宗元
      <BR>
    <P>
      <center>
             <INPUT TYPE = "submit" value = "提交答案" name = "submit">
      </center>
  </FORM>
                      <!-- 例程 4-4 defen.jsp -->
<%@ page contentType = "text/html;charset = GB2312" %>
<HTML>
 <BODY>
  <FONT size = 3>
  <%
      int n = 0;
      String s1 = request.getParameter("r");          //获取参数值
      String s2 = request.getParameter("P");
      String s3 = request.getParameter("q");
      if(s1 == null)
        {s1 = "";}
      if(s2 == null)
        {s2 = "";}
```

```
            if(s3 == null)
              {s3 = "";}
            if(s1.equals("d"))
              { n++ ;}
            if(s2.equals("c"))
              { n++ ;}
            if(s3.equals("a"))
              { n++ ;}
           %>
          <P>您好,您本次的得分是: <% = n %>分
      </FONT>
    </BODY>
  </HTML>
```

程序 danxuan.jsp 的运行界面如图 4-3 所示。

当选择了相应选项后,页面会转到 defen.jsp 页面执行,如全部选择了正确答案后,显示如图 4-4 所示的结果。本例源代码存放于本书配套素材中的 danxuan.jsp 文件和 defen.jsp 文件中(文件路径为 code\4)。

图 4-3　danxuan.jsp 的输出界面

图 4-4　defen.jsp 的统计结果

专家点拨: 以上两例都是利用 request 对象的 getParameter 方法得到表单元素的值,注意 getParameter 方法的参数名一定要与对应的表单元素名字相同。如在例 4-1 的表单中信息框为<input type="text" name="boy">,它的名字是 boy,对应的 request 对象的 getParameter 方法就应该是 request.getParameter("boy")。

【例 4-3】 获取服务器端的有关信息。

在本例的 request.jsp 文件中,利用 request 对象的各种方法来获取服务器端的各种信息参数,并将其显示出来。

本例源代码如下:

```
                    <!-- 例程 4 - 5 request.jsp -->
<% @page contentType = "text/html;charset = GB2312" %>
<html>
  <body>
   <h2 align = center > Request 对象获得服务器端参数 </h1 >
    <font size = "3" color = "blue">
        request.getMethod(): <% = request.getMethod() %>  <br>
        request.getProtocol(): <% = request.getProtocol() %><br>
        request.getServletPath(): <% = request.getServletPath() %><br>
        request.getServerName(): <% = request.getServerName() %><br>
```

```
            request.getServerPort(): <%= request.getServerPort() %><br>
            request.getRemoteUser(): <%= request.getRemoteUser() %><br>
            request.getRemoteAddr(): <%= request.getRemoteAddr() %><br>
            request.getRemoteHost(): <%= request.getRemoteHost() %><br>
            <hr>
              正在使用的浏览器: <%= request.getHeader("User-Agent") %>
            <hr>
        </font>
      </body>
    </html>
```

程序运行后在网页上输出的结果如图 4-5 所示。本例源代码存放于本书配套素材中的 request.jsp 文件中(文件路径为 code\4)。

Request对象获得服务器端参数

```
request.getMethod(): GET
request.getProtocol(): HTTP/1.1
request.getServletPath(): /request.jsp
request.getServerName(): localhost
request.getServerPort(): 8080
request.getRemoteUser(): null
request.getRemoteAddr(): 127.0.0.1
request.getRemoteHost(): 127.0.0.1
-------------------------------------------
正在使用的浏览器: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT
5.1; Trident/4.0; QQDownload 570; TencentTraveler 4.0; Embedded
Web Browser from: http://bsalsa.com/; .NET CLR 1.1.4322; .NET
CLR 2.0.50727)
```

图 4-5 request.jsp 的输出结果

4.3 response 对象

response 对象封装了 JSP 的响应,其主要功能是将服务器处理后的结果传回到客户端,以响应客户端的请求,由于 JSP 中有 out 对象可以方便地向客户端输出内容,所以 response 对象常用于与 Cookie 有关的操作及网页的重定向。

客户访问服务器使用的是 HTTP 请求包,系统将 HTTP 请求包封装为 request 对象。而服务器响应客户,即向客户发送信息时使用的是 HTTP 响应包,系统也同样将 HTTP 响应包封装为 response 对象。在 JSP 页面中,可以使用 response 对象的方法动态控制响应方式,向客户端发送数据。

4.3.1 HTTP 响应包

HTTP 响应包与 HTTP 请求包结构类似,一个 HTTP 响应包由 3 个部分组成:一个状态行、多个响应头和信息体。

1. 状态行

描述服务器处理 HTTP 请求是否成功。例如,是否收到请求包、请求被拒绝、请求超时、服务器发生错误等。

2. 响应头

HTTP 响应包发送的目标地址(IP)。

3. 信息体

发送到服务器端的正文。例如,在客户端显示的信息。

服务器响应客户时,它发送到客户端的首行被称为状态行。状态行由 3 位数的状态代码和描述状态代码的文字组成。下面是对状态代码的分类描述:

1xx:1 开头的 3 位数字,主要用于实验。

2xx:2 开头的 3 位数字,表明客户端的请求已成功。

3xx:3 开头的 3 位数字,表明处理客户的请求以前,应做一些处理。

4xx:4 开头的 3 位数字,表明浏览器请求是非法的或无效的。

5xx:5 开头的 3 位数字,表明服务器出现了问题。

一般不需要修改状态行,在出现问题时,服务器会自动响应,发送相应的状态代码到客户端,也可以使用 setStatus(int n)方法来增加状态行的内容。

表 4-1 给出了服务器响应客户时发送到客户端的状态代码描述。

表 4-1 状态代码表

状态代码	代 码 说 明
101	服务器正在升级协议
100	客户可以继续
201	请求成功且在服务器上创建了新的资源
202	请求已被接受但还没有处理完毕
200	请求成功
203	客户端给出的原信息不是发自服务器的
204	请求成功,但没有新信息
205	客户必须重置文档视图
206	服务器执行了部分 get 请求
300	请求的资源有多种表示
301	资源已经被永久移动到新位置
302	资源已经被临时移动到新位置
303	应答可以在另外一个 URL 中找到
304	Get 方式请求不可用
305	请求必须通过代理来访问
400	请求由语法错误
401	请求需要 HTTP 认证
403	取得了请求但拒绝服务
404	请求的资源不可用
405	请求所用的方法是不允许的
406	请求的资源只能用请求不能接受的内容特性来响应
407	客户必须得到认证
408	请求超时

续表

状态代码	代 码 说 明
409	发生冲突,请求不能完成
410	请求的资源已经不可用
411	请求需要一个定义的内容长度才能处理
413	请求太大,被拒绝
414	请求的 URL 太大
415	请求的格式被拒绝
500	服务器发生内部错误,不能服务
501	不支持请求的部分功能
502	从代理和网关接受了不合法的字符
503	HTTP 服务暂时不可用
504	服务器在等待代理服务器应答时发生超时
505	不支持请求的 HTTP 版本

4.3.2　response 对象的常用方法

response 对象可以使用的常用方法如下：

(1) addHeader(String name,String value)：添加一个具有指定名称和值的响应标头，即 HTTP 头文件。Header 将会传到客户端，如果有同名的 Header 存在，那么原来的 Header 会被覆盖。

(2) setHeader(String name,String value)：设定一个具有指定名称的 HTTP 文件头的值，如果该值已经存在，那么将会用新的值去改写原来的值。

(3) addCookie(Cookie cookie)：添加一个 Cookie 对象，用来保存客户端的用户信息，可以用 request 对象的 getCookies()方法获得 Cookie。

(4) containsHeader(String name)：判断指定名字的 HTTP 文件头是否已经设置，返回一个布尔值。

(5) sendError(int sc)：使用指定的状态向客户端发送一个错误信息。如“505 指示服务器内部错误”,“404 指示网页找不到的错误”等。

(6) SendRedirect(URL)：把响应发送到另一个指定的页面(URL)进行处理，即重定向页面。

(7) flushBuffer()：强制将当前缓冲区的内容发送到客户端。

(8) setContenType(String type)：设置被发送到客户端响应的内容类型。

(9) setLocale(Locale:locale)：设置响应的语言环境。

(10) setStatus(int:statuscode)：设置响应的状态行。

(11) isCommitted()：判断响应是否已经提交，返回一个布尔值。

(12) reset()：清除缓冲区中已经存在的数据，同时也清除状态码和标头。

4.3.3　response 对象应用实例

【例 4-4】　控制网页的刷新频率，在页面中，实时显示当前的时间。

要实时显示当前的时间，必须每秒钟刷新一次页面，这就要求向客户输出的响应包中必

须含有响应头 Refresh，其值为 1，单位是秒。本例利用 response 对象的 setHeader()方法添加响应头和属性值。

本例源代码如下：

```
                    <!-- 例程 4-6 refresh.jsp -->
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.util.*" %>
<HTML>
  <BODY>
   <center>
    <FONT size=4 color=red>
     <p>现在的时间是:<br>
      <%= new java.util.Date() %>
      <%
        response.setHeader("Refresh","1");
            //添加一个 Refresh 响应头，并将其值设为 1，使客户端每隔 1 秒刷新该页面
       %>
    </FONT>
   </center>
  </BODY>
</HTML>
```

本例在网页上的显示结果如图 4-6 所示。用户运行后可以看到这个时间在不断地变化，每一秒钟刷新一次。本例源代码存放于本书配套素材中的 refresh.jsp 文件中（文件路径为 code\4）。

现在的时间是:
Wed Oct 07 11:28:25 CST 2009

图 4-6 refresh.jsp 的显示结果

【例 4-5】 改变浏览器的输出类型，向客户端输出 txt 类型的文件。

这是一个应用 setContenType()方法的例子，用来动态设置文件的输出类型。实际上是要求设置 page 指令的 contentType 属性值为 application/mstxt。当用户访问一个 JSP 页面时，如果该页面用 page 指令设置页面的 contentType 属性是 text/html，那么 JSP 引擎将按照这种属性值做出反映。如果要动态改变这个属性值来响应用户，就需要使用 setContenType()方法来改变 contentType 属性值。

本例源代码如下：

```
                    <!-- 例程 4-7 type.jsp -->
<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
 <BODY>
  <FONT size=3>
   <P>学习 response 对象的 setContentType 方法的使用<BR>
   <P>要将当前页面保存为.txt 文件吗?
    <FORM action="" method="get"  name="form1">
     <INPUT TYPE="submit"  name="Y"  value="是">
    </FORM>
  <%
      String str=request.getParameter("Y");
      if(str==null)
        {
```

```
          str = " ";
        }
      byte b[ ] = str.getBytes("ISO - 8859 - 1");
      str = new String(b);

      if(str.equals("是"))
        {
          response.setContentType("application/mstxt;charset = GB2312");
        }
   %>
  </FONT>
 </BODY>
</HTML>
```

程序运行后在网页上输出的结果如图 4-7 所示。单击“是”按钮后会弹出“保存”对话框来保存该页面。本例源代码存放于本书配套素材中的 type.jsp 文件中(文件路径为 code\4)。



图 4-7　type.jsp 的输出结果

【例 4-6】 实现从一个网页到另一个网页的重定向。

本程序由两个 JSP 页面文件构成,send.jsp 页面向 redirect.jsp 页面提供姓名信息。主要使用 SendRedirect(URL)方法来实现网页的动态重定向。

本例源代码如下:

```
                    <!-- 例程 4 - 8 send.jsp -->
<% @ page contentType = "text/html;charset = GB2312"  %>
<HTML>
 <BODY>
  <P>请填写姓名:<BR>
   <FORM action = "redirect.jsp" method = "get" name = form>
     <INPUT TYPE = "text" name = "xm" size = "20">
     <P><INPUT TYPE = "submit" value = "提交">
   </FORM>
 </BODY>
</HTML>
                    <!-- 例程 4 - 9 redirect.jsp -->
<% @ page contentType = "text/html;charset = GB2312"  %>
<HTML>
 <BODY>
  <%
     String str = null;
     str = request.getParameter("xm");
     if(str == null)
       {
         str = "";
       }
     byte b[ ] = str.getBytes("ISO - 8859 - 1");
     str = new String(b);
     if(str.equals(""))
       {
```

```
            response.sendRedirect("send.jsp");
        }
      else
        {
            out.println(str + ":");        //输出
            out.println("欢迎您的到来!");
        }
    %>
</BODY>
</HTML>
```

send.jsp 运行后，在网页的输出结果如图 4-8 所示。

在该页面的文本框中输入姓名（如刘三姐）后，单击“提交”按钮，则页面被重定向到 redirect.jsp 页面，输出结果如图 4-9 所示。本例源代码存放于本书配套素材中的 send.jsp 文件和 redirect.jsp 文件中（文件路径为 code\4）。



图 4-8　send.jsp 的输出结果

刘三姐：欢迎您的到来！

图 4-9　重定向到 redirect.jsp 的输出结果

4.4　session 对象

session 是一种服务器单独处理与记录客户端使用者信息的技术。众所周知，客户与服务器的通信是通过 HTTP 协议完成的。但 HTTP 协议是一种无状态协议。即一个客户向服务器发送请求（request），然后服务器返回响应（response），连接就关闭了。服务器端不保留客户与服务器每一次连接的信息，因此，服务器无法判断上下两次连接是否是同一个客户。要想记住客户的连接信息，必须使用会话对象（session）。

session 对象是 JSP 中十分重要的一个对象，用来记录每个客户端的访问状态，并跟踪客户端的操作状态，一般来说，不同的用户所对应的 session 对象是不同的。

4.4.1　会话及相关概念

1. 会话

从一个客户打开浏览器连接到服务器的某个服务目录，再到客户关闭浏览器，该过程称为一个会话（其间，客户访问的是同一个 Web 目录中的网页）。这时，在服务器端，系统为该客户创建了一个 session 对象，在客户端，系统为该客户创建了一个 Cookie 对象。一个客户对同一个服务目录中不同网页的访问属于同一个会话。

当一个客户首次访问服务目录中的一个 JSP 页面时，JSP 引擎为该客户创建一个 session 对象，并设定其中的内容。这些 session 都是独立的，服务器端可以借此来辨别使用者的信息进而提供独立的服务。同一个客户访问的服务目录不同，JSP 引擎为该客户创建

不同的 session 对象。

2. session 对象与 Cookie 对象

当 JSP 引擎为客户创建一个 session 对象后，session 对象被分配了一个 String 类型的 ID 号，JSP 引擎同时将此 ID 号发送到客户端，并存放在 Cookie 中。这样，代表同一个客户会话的 session 对象和 Cookie 对象建立了一一对应关系。即每一个客户会话信息保存在 session/Cookie 对象中。当用户再次访问连接该服务器的其他页面时，就不再分配给用户新的 session 对象，直到关闭浏览器，服务器端该用户的 session 对象才取消，并且和用户的对应关系也取消。如果重新打开浏览器再连接到该服务器时，服务器为用户再创建一个新的 session 对象。

session 对象与 Cookie 对象相比较，有以下的区别。

(1) 从存储位置看，session 对象保存在 Web 服务器的内存中，Cookie 对象则保存在客户端计算机的硬盘上。

(2) 从存活时间看，session 对象是随用户连接服务器而临时生成的，当用户关闭浏览器或会话失效时，session 对象会随之消失；Cookie 对象随服务器响应而保存在客户端，其失效期限可以用 setMaxAge()方法设置，可以长期存储在客户端。

(3) 从安全性看，session 对象保存在 Web 服务器上，安全性高，用户不能修改，而且关闭浏览器后失效，但这会对服务器性能产生一定影响；Cookie 对象则保存在客户端，有可能被用户删除，安全性比较差。

(4) 从实现方式看，session 对象是一个动作连续的状态，是一个浏览器与服务器的交互会话过程，它在服务器中持续存在直到不用为止；Cookie 对象可以在后续的请求中由客户端发送到服务器，从而确定用户的身份，这对同一个用户反复访问同一个网站时保存信息是十分有用的。

3. session 对象与线程

当多个客户点击同一个页面时，JSP 引擎为每个客户启动一个线程，也就是说，一个客户对应一个线程，每个线程对应一个 session 对象，每个线程的 session 对象不同。

4. session 对象的生命周期

从一个客户会话开始到会话结束这段时间称为 session 对象的生命周期。具体上说，指客户访问某 Web 目录下的页面到关闭浏览器，并离开该 Web 目录，这段时间称为 session 对象的生命周期。

4.4.2 session 对象的常用方法

session 对象可以使用的常用方法如下：

(1) getAttribute(String name)：获得指定名字的属性，如果该属性不存在，将会返回 null。

(2) getAttributeNames()：返回 session 对象中存储的每一个属性对象，结果集是一个 Enumeration 类的实例。

(3) getCreationTime()：返回 session 对象被创建的时间，单位为毫秒。从格林尼治时间 1970 年 1 月 1 日零时算起。

(4) getId()：返回 session 对象在服务器端的 ID 编号。每生成一个 session 对象，服务器都会给它一个编号，而且这个编号不会重复，这样服务器才能根据编号来识别 session 对象，并且正确地处理某一特定的 session 对象及其提供的服务。

(5) getLastAccessedTime()：返回当前 session 对象最后一次被操作的时间，单位为毫秒。从格林尼治时间 1970 年 1 月 1 日零时算起。

(6) getMaxInactiveInterval()：获取 session 对象的最大生存时间，若超过这个时间，session 对象就会失效，单位为秒。

(7) setMaxInactiveInterval (int interval)：设置 session 对象的有效时间(超时时间)，单位为秒。

专家点拨：在网站的实际应用中，30 分钟的有效时间对某些网站来说有些太短，但对有些网站来说又有些太长。因此，为了减少服务器资源的浪费，就应该设置相应的有效时间。若设置为负数表示永远不会超时。

(8) removeAttribute(String name)：删除指定属性的属性值和属性名。如果在有效的时间内，用户做出了新的请求，那么服务器就会将其看做是一个新的用户，此时，服务器将创建一个新的 session 对象，而原来旧的 session 对象信息则会消失。

专家点拨：在设定的有效时间里，只有当用户退出或当前页面处理完后，removeAttribute (String name)方法才会起作用。

(9) setAttribute(String name,Java.lang.Object value)：设定指定名字的属性，并且把它存储在 session 对象中。

(10) invalidate()：注销当前的 session 对象，并将 session 对象存放的内容完全抛弃。

(11) isNew()：判断 session 对象是否为“新”的，所谓“新”的 session 对象，即指 session 对象已由服务器产生，但是客户端尚未使用，返回一个布尔值。

4.4.3 session 对象应用实例

【例 4-7】 获取访问页面的次数。

在本例中调用了 setAttribute()方法和 getAttribute()方法，使用一个变量 num 来记录访问页面的次数，每访问一次，num 的值都会加 1。

本例源代码如下：

```
                  <!-- 例程 4-10 cishu.jsp -->
<%@ page language="java" contentType="text/html; charset=gb2312" %>
<html>
 <head>
  <title>session 用法实例</title>
 </head>
  <body bgcolor=#ccffcc>
   <font size=5>
    <%
      int num = 0;
```

```
        Object ob = session.getAttribute("num");  //从 session 对象中取得 num 的值
        if(ob == null)
         {
           session.setAttribute("num",String.valueOf(num));
                                                       //设定 session 对象的 num 变量的值
         }
        else
         {
           num = Integer.parseInt(ob.toString());
           num + = 1;   //来访人数加 1
           session.setAttribute("num",String.valueOf(num));
         }
       %>
       这是第 <% = num %> 次访问该页面
    </font>
   </body>
</html>
```

该程序运行后，每访问一次，变量 num 的值都会加 1，如第 10 次运行该页面时的效果如图 4-10 所示。本例源代码存放于本书配套素材中的 cishu.jsp 文件中(文件路径为 code\4)。

session用法实例

这是第 10 次访问该页面

图 4-10 第 10 次访问 cishu.jsp 的效果

【例 4-8】 显示客户会话的 ID 号。

本例的功能是同一个客户访问两个不同的 Web 目录中的页面，查看该客户在不同的 Web 目录中 session 对象的 ID 号。

本例源代码如下：

```
                  <!-- 例程 4 - 11 id.jsp -->
<%@ page contentType = "text/html;charset = GB2312" %>
<HTML>
 <BODY>
  <P>
  <%
     String s = session.getId();
     String str = response.encodeURL("liu.jsp");
   %>
  <P>您在访问 id.jsp 页面<br><br>
     您的 session 对象的 ID 是：<br><br>
    <% = s %><br><br>
    <FORM action = "<% = str %>" method = post  name = form>
      <INPUT TYPE = "submit" value = "转向 liu 页面"  name = submit>
    </FORM>
    <FORM action = "../4/file.jsp" method = post name = form>
      <INPUT TYPE = "submit" value = "转向 file 页面" name = submit>
    </FORM>
</BODY>
</HTML>

                  <!-- 例程 4 - 12 liu.jsp -->
<%@ page contentType = "text/html;charset = GB2312" %>
```

```
<HTML>
 <BODY>
  <%
    String s = session.getId();
    String str = response.encodeURL("wang.jsp");
   %>
  <P>您在访问 liu 页面 <br><br>
      您的 session 对象的 ID 是: <br><br>
    <% = s %>
  <p>单击超链接,链接到 wang 页面 <br><br>
  <A HREF = "<% = str %>"> 欢迎到 wang 页面来!</A>
</BODY>
</HTML>
                                    <!-- 例程 4-13 wang.jsp -->
<%@ page contentType = "text/html;charset = GB2312"  %>
<HTML>
 <BODY>
  <%
    String s = session.getId();
    String str = response.encodeURL("id.jsp");
   %>
  <P>您在访问 wang 页面 <br><br>
      您的 session 对象的 ID 是: <br><br>
      <% = s %>
  <p>单击超链接,链接到 ID 页面 <br><br>
  <A HREF = "<% = str %>">   欢迎到 ID 页面来!   </A>
</BODY>
</HTML>

                                    <!-- 例程 4-14 file.jsp -->
<%@ page contentType = "text/html;charset = GB2312"  %>
<HTML>
<BODY>
  <%
    String s = session.getId();
   %>
  <P>您在访问 file 页面 <br><br>
      您的 session 对象的 ID 是: <br><br>
     <% = s %>
  <p>单击超链接,返回到 ID 页面<br><br>
  <a href  = "../4/id.jsp">   欢迎到 ID 页面来!   </a>
 </BODY>
</HTML>
```

【说明】

本程序由 4 个 JSP 页面文件构成,分别是 id.jsp、liu.jsp、wang.jsp 和 file.jsp,4 个页面的交互关系如图 4-11 所示。

(1) id.jsp 文件:用于获取访问本页面的客户的 session 对象的 ID 号并输出 ID 号。在该页面中创建表单,该表单提交时,将信息提交给 liu.jsp 页面或是 file.jsp 页面。该页面在网页上的运行效果如图 4-12 所示。

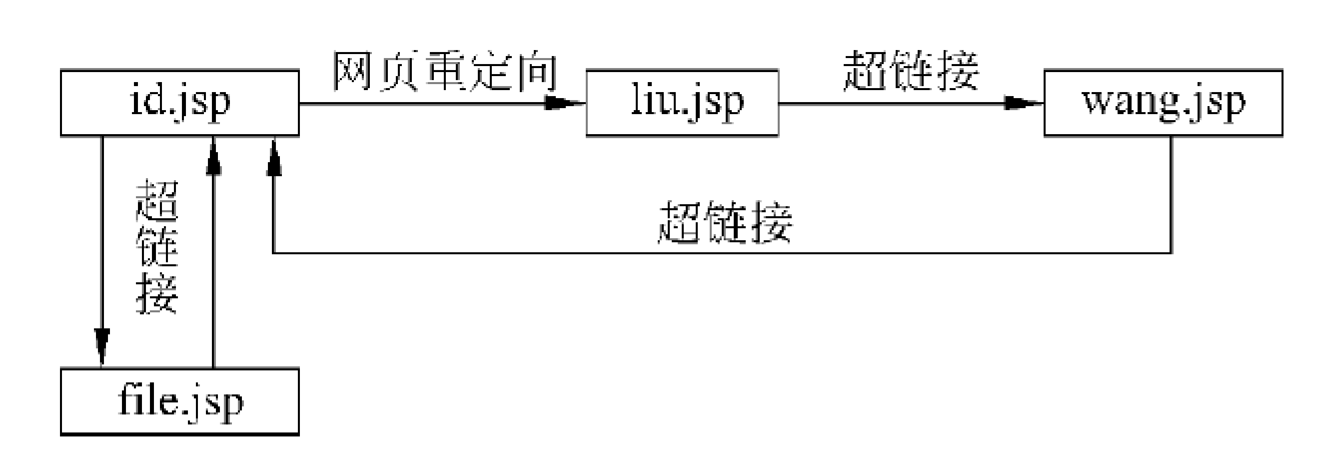


图 4-11　页面交互关系图

（2）liu.jsp 文件：单击 id.jsp 页面中的“转向 liu 页面”按钮可进入到该页面，该页面用于获取访问本页面的客户的 session 对象 ID 号并输出 ID 号，同时设置链接到 wang.jsp 页面的超链接。该页面在网页上的运行效果如图 4-13 所示。

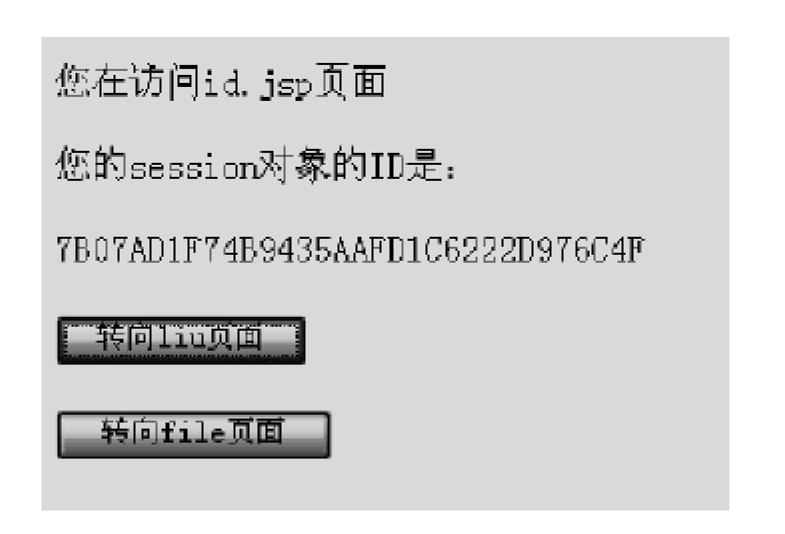


图 4-12　id.jsp 的输出效果

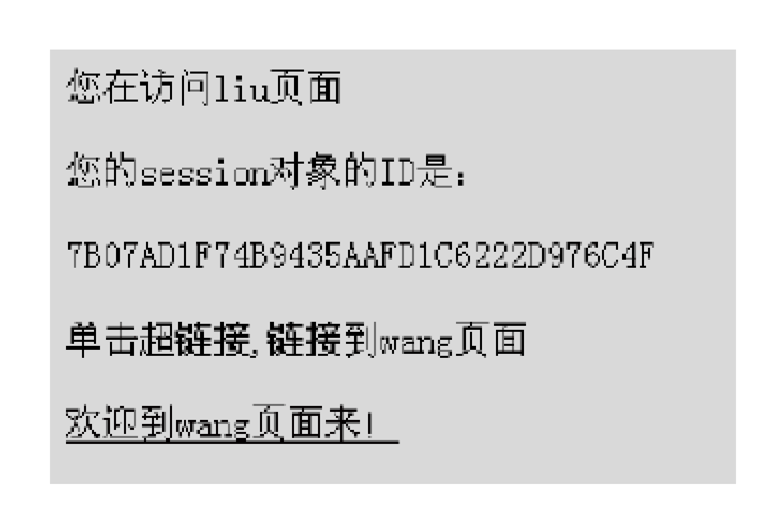


图 4-13　liu.jsp 的输出效果

（3）wang.jsp 文件：单击超链接“欢迎到 wang 页面来！”可进入到该文件的输出页面，该文件用于获取和输出访问本页面的客户的 session 对象 ID 号，同时设置链接到 id.jsp 页面的超链接。该页面在网页上的运行效果如图 4-14 所示。

（4）file.jsp 文件：在 id.jsp 页面中单击“转向 file 页面”按钮，可进入到该文件的输出页面，该文件用于获取和输出访问本页面的客户的 session 对象 ID 号，同时设置链接到 id.jsp 页面的超链接。该页面在网页上的运行效果如图 4-15 所示。

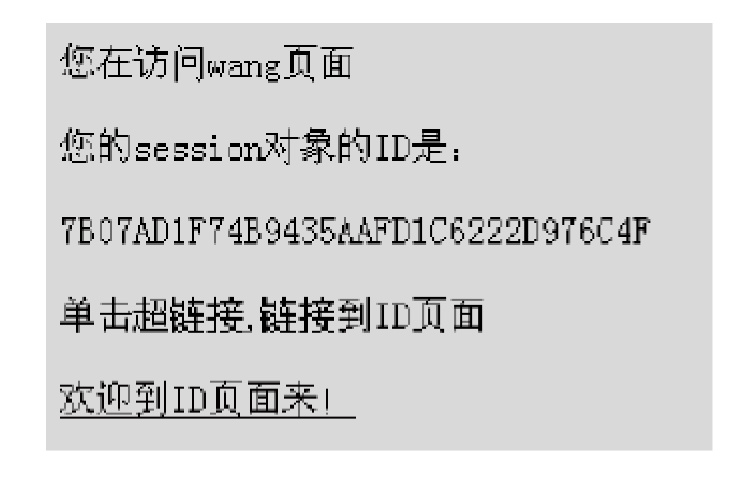


图 4-14　wang.jsp 的输出效果

您在访问file页面
您的session对象的ID是：
7B07AD1F74B9435AAFD1C6222D976C4F
单击超链接，返回到ID页面
欢迎到ID页面来！

图 4-15　file.jsp 的输出效果

本例源代码存放于本书配套素材的 file.jsp 文件中（文件路径为 code\4）。

4.5　application 对象

application 对象的主要作用是为多个应用程序保存信息，直到服务器关闭为止。当多个客户点击同一个页面时，JSP 引擎会为每个客户启动一个线程，这些线程共享同一个 application 对象。由于所有客户共享同一个 application 对象，任何客户对 application 对象

中数据的改变都会影响到其他的客户，因此，对该对象的操作需要实现同步处理。

从上一节知识已经知道，不同的客户与服务器连接时有不同的 session 对象，同一个客户浏览同一个服务器的不同目录时，有不同的 session 对象。而与 session 对象不同的是，application 对象在服务器启动后就产生了，在 JSP 服务器运行时，仅有一个 application 对象。所有客户的 application 对象是相同的，即所有客户共享这个内置的 application 对象。它由服务器创建，也由服务器自动清除，不能被用户创建和删除。

application 对象随着服务器启动而创建，随着服务器关闭而消失。application 对象的生命周期指从 application 对象创建到服务器关闭这段时间。

4.5.1 application 对象的常用方法

在使用 session 对象时，各个客户端共享一个 session 对象，而使用 application 对象时，在同一个服务器中的 JSP 文件共享一个 application 对象。

application 对象的主要方法有如下几种。

(1) getAttribute(String name)：返回由 name 指定名字的 application 对象的属性值。返回值是一个 Object 对象，如果没有，则返回 null。

(2) getAttributeNames()：返回所有 application 对象属性的名字，结果集是一个 Enumeration 类型的实例。

(3) getInitParameter(String name)：返回由 name 指定名字的 application 对象的某个属性的初始值，如果没有参数，就返回 null。

(4) getServerInfo()：返回 Servlet 编译器当前版本的信息。

(5) setAttribute(String name, Object obj)：将参数 Object 指定的对象 object 添加到 application 对象中，并为添加的对象指定一个属性。

(6) removeAttribute(String name)：删除一个指定的属性。

(7) getContext(String urlpath)：返回一个对应于指定 URL 的 ServletContext 对象，其中参数 urlpath 表示一个站点的相对路径。

(8) getMimeType(String filename)：返回指定文件的 MIME 类型，若 MIME 类型未知，则返回 null。

(9) getRealPath(String path)：返回一个字符串，其中包含指定上下文相对路径的文件系统路径。若 Web 容器不能将该路径转换为文件系统路径，则返回 null。

4.5.2 application 对象应用实例

【例 4-9】 使用 application 对象记录页面的访问次数。

本例用于记录一个页面总共被访问了多少次，当用户访问这个页面时，就会将 application 对象中的 num 值加 1，它的生命周期为从服务器启动开始到服务器关闭为止。

本例源代码如下：

```
<!-- 例程 4-15 application.jsp -->
<%@ page import="java.util.*"%>
<%@ page language="java" contentType="text/html;charset=gb2312"%>
```

```
<html>
  <body>
    <%
        int num;
        if(application.getAttribute("num") == null)
          {
        application.setAttribute("num","1");
          }
        else
        {
        num = Integer.parseInt((String)application.getAttribute("num"));
        num++;
        application.setAttribute("num",Integer.toString(num));
        }
     %>
    <h3>该页面已经被浏览了<% = (String)application.getAttribute("num") %>次</h>
  </body>
</html>
```

读者可比较本例和上一节的例4-7,这两个程序都是记录页面被访问的次数,只是本例用的是application对象记录,而例4-7用的是session对象记录。请思考这两个程序在运行过程中会有什么样的差别(提示:可以分别用多个浏览器访问这两个程序,看看有什么不同)。本例源代码存放于本书配套素材中的application.jsp文件中(文件路径为code\4)。

【例4-10】 读取系统信息。

本例主要是输出页面所在的实际路径、使用的JSP引擎和application对象对应的字符串。

本例源代码如下:

```
                          <!-- 例程 4-16 sysinfo.jsp -->
<%@ page contentType = "text/html;charset = GB2312"  %>
<html>
 <body>
  <center>
   <h2>系统信息</h2>
   <% String  path = "/sysinfo.jsp"; %>
    <p>context 数据的内容: <% = application.getContext(path) %>
                                         //读取 path 路径中的 ServletContext
    <p>文件的格式: <% = application.getMimeType(path)  %>
    <p>本页面文件的实际路径:<% = application.getRealPath(path) %>
                                         //通过相对路径获得实际路径
    <p>jsp 引擎:<% = getServletInfo() %>          //当前 JSP 引擎
    <p>application 对象 ID:
    <p><% = getServletContext() %>
  </center>
 </body>
</html>
```

本程序在网页上的输出结果如图4-16所示。本例源代码存放于本书配套素材中的sysinfo.jsp文件中(文件路径为code\4)。

系统信息

context数据的内容：null

文件的格式：null

本页面文件的实际路径:E:\code\4\sysinfo.jsp

jsp引擎:Jasper JSP 2.1 Engine

application对象ID:

org.apache.catalina.core.ApplicationContextFacade@18a8ce2

图 4-16　sysinfo.jsp 的输出结果

4.6　其他内部对象

除了以上介绍的 4 个内置对象外，JSP 中还有其他一些对象，如 out、page、config 等。由于篇幅所限，本节将对这些对象作简单的介绍。

4.6.1　out 对象

out 对象是一个输出流，是 JSP 使用最频繁的对象，能把结果输出到网页上。并且 out 对象还管理应用服务器上的输出缓冲区。

out 对象的常用方法如下：

(1) out.print()或 out.println()：输出指定的字符串或者 HTML 标签。其中 out.println()方法会在输出的数据后自动加上一个换行的符号。

专家点拨：out.println()方法输出的换行只是在输出 html 代码时有空行，在浏览器解析时这个空行将被忽略，要想在页面中实现换行，需要通过 out.println("
")方法来实现。

(2) out.newLine()：输出一个换行符号。

(3) out.clearBuffer()：清除缓冲区中的数据，并且把数据写到客户端。

(4) out.clear()：清除缓冲区中的数据，但不把数据写到客户端。

(5) out.flush()：输出缓冲区中的数据并清除。

(6) out.getBufferSize()：获取缓冲区的大小。缓冲区大小可以用<%@page buffer="size"%>语句来设置。

(7) out.getRemaining()：获取缓冲区剩余空间的大小。

(8) out.isAutoFlush()：返回布尔值，如果自动缓冲，则返回 true，否则返回 false。是否自动缓冲可以用<%@page isAutoFlush()= "true/false" %>语句来设置。

【例 4-11】　在客户端输出一个表格。

程序源代码如下：

```
                <!-- 例程 4-17 table.jsp -->
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.util.*" %>
```

```
<HTML>
 <BODY>
  <Center>
  <%
     out.println("<FONT face=宋体 size=2>");
     out.println("<Table Border>");
     out.println("<TR>");
     out.println("<TH width=80>" + "姓名" + "</TH>");
     out.println("<TH width=80>" + "性别" + "</TH>");
     out.println("<TH width=80>" + "年龄" + "</TH>");
     out.println("</TR>");
     out.println("<TR>");
     out.println("<TD align=center>" + "张三丰" + "</TD>");
     out.println("<TD align=center>" + "男" + "</TD>");
     out.println("<TD align=center>" + "50" + "</TD>");
     out.println("</TR>");
     out.println("</Table>");
     out.println("</FONT>");
   %>
  </Center>
 </BODY>
</HTML>
```

程序的输出结果如图 4-17 所示。本例源代码存放于本书配套素材中的 table.jsp 文件中(文件路径为 code\4)。

姓名	性别	年龄
张三丰	男	50

图 4-17 table.jsp 的输出结果

4.6.2 page 对象

page 对象属于 java.lang.Object 类型,它是处理当前请求的 JSP 实现类的实例。page 对象指向当前 JSP 页面本身,更确切地说,它代表 JSP 被转译后的 Servlet,因此,它可以调用 Servlet 类所定义的方法,在程序中可以用 this 来引用它。

【例 4-12】 输出 JSP 页面对象的 ID 号和 hash 代码值。

本例调用 page 对象的 hashCode()方法和 toString()方法,分别获取 page 对象的 hash 代码值和 ID 号。程序源代码如下:

```
                <!-- 例程 4-18 page.jsp -->
<%@ page contentType="text/html;charset=GB2312" %>
<html>
<body>
  <%
     int hashCode=page.hashCode();
     String thisStr=page.toString();
     out.println("page 对象的 ID 值: " + thisStr+"<br>");
     out.println("page 对象的 hash 代码: " + hashCode+"<br>");
   %>
</body>
</html>
```

程序的输出效果如图 4-18 所示。本例源代码存放于本书配套素材中的 page.jsp 文件

中(文件路径为 code\4)。

```
page对象的ID值: org.apache.jsp.page_jsp@1f8bd0d
page对象的hash代码: 33073541
```

图 4-18　page.jsp 的输出效果

4.6.3　pageContext 对象

pageContext 是 javax.servlet.jsp.PageContext 类的一个实例。pageContext 对象相当于 JSP 页面所有功能的集成者,它提供了对 JSP 页面内所有对象及命名空间的访问。使用该对象可以访问当前页面所在 session 的属性值,也可以访问当前页面所在 application 的属性值,并且允许向其他应用组件转发 Request 对象,或者从其他应用组件包含 Request 对象。

pageContext 对象的常用方法如下。

(1) getAttribute():返回与指定范围内名称有关的变量或 null。

(2) forward(String relativeUrlPath):把页面重定向到另一个页面或者 Servlet 组件上。

(3) findAttribute():用来按照页面请求、会话以及应用程序范围的顺序实现对某个已经命名属性的搜索。

(4) getexception():返回当前的 exception 对象。

(5) setAttribute():用来设置默认页面的范围或者指定范围中的已命名对象。

(6) removeAttribute():用来删除默认页面范围或指定范围中已命名的对象。

4.6.4　config 对象

config 对象是 JSP 页面通过 JSP Container 进行初始化时被传递的对象。config 对象具有 Web 服务器环境设定值。

使用 config 对象的优点是在修改需要在 Web 服务器中处理的变量时,不需要逐一修改 JSP 文件,只要修改相应属性文件的内容就可以了。

config 对象的常用方法如下:

(1) getInitParameter(String name):返回指定初始参数的内容。返回值为 String 类型。

(2) getInitParameterNames():返回所有初始参数的名称。返回值为 Enumeration 类型。

(3) getServletNames():返回 Servlet 的名称。返回值为 String 类型。

(4) getServletContext():返回 Servlet 属于哪一个 application。

4.6.5　exception 对象

exception 对象用来处理 JSP 文件在执行时所发生的错误和异常。exception 对象可以配合 page 指令一起使用,通过指定某一个页面为错误处理页面,把所有的错误都集中到那

个页面进行处理。这样可以使得整个系统更加健壮，也使得程序的流程更加清晰，这也是JSP比ASP和PHP先进的地方。exception对象的常用方法如下：

（1）getMessage()：返回错误信息。

（2）printStackTrace()：为标准错误的形式输出一个错误和错误的堆栈。

（3）toString()：以字符串的形式返回一个对异常的描述。

专家点拨： *必须在isErrorPage=true的情况下才可以使用exception对象。*

4.7　JSP程序的调试

编写程序不可能没有一点错误，在JSP程序的开发过程中，调试程序是一个非常重要、必不可少的环节。通过调试能够发现程序中存在的各种错误，从而及时改正错误。JSP程序中包括HTML标签和Java代码，其中Java代码在服务器端进行编译和执行，这也增加了调试JSP程序的难度。

4.7.1　三种错误类型

当程序不能正常运行或运行结果不正确时，就表明程序中有错。在JSP程序中，常见的错误有以下3种：

1．语法错误

语法错误又称为编译错误，也就是编写的语句不符合语法规范。当JSP程序中存在语法错误时，JSP文件在服务器端被JSP引擎编译成Servlet时，就会发生错误，无法通过编译。

2．运行错误

当JSP引擎将JSP文件编译成Servlet加载到内存执行时发生的错误称为运行错误，运行错误会被Java的异常处理机制处理。

3．逻辑错误

JSP程序可以被运行，但运行结果却与期望值不符，这是由于某种原因JSP程序内部逻辑发生了错误，这种错误不返回错误信息，是最难调试的一种。

4.7.2　JSP语法错误的调试

本节以一个乘法九九表为例来分析在编写JSP程序时经常会遇到的语法错误。

【例4-13】 输出乘法九九表。

本例源代码如下：

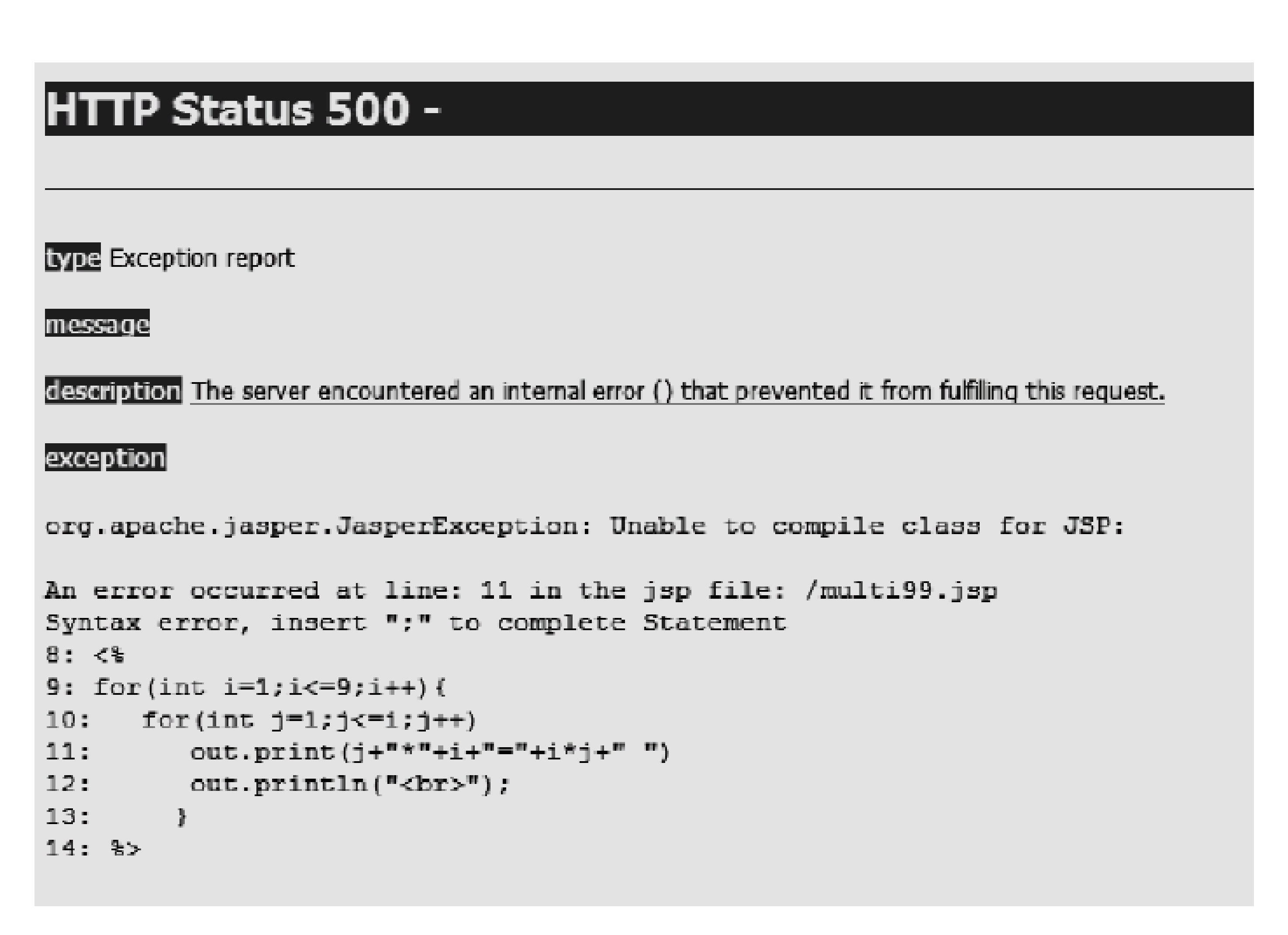
```
<!-- 例程 4 - 19 multi99.jsp -->
<html>
<head>
  <title>JSP 脚本代码</title>
```

```
4    </head>
5      <body>
6        <%@ page contentType = "text/html; charset = gb2312" %>
7      <font color = "red" >
8      <%
9        for(int i = 1;i <= 9;i ++ ){
10         for(j = 1;j <= i;j ++ )
11           out.print(j + " * " + i + " = " + i * j + " ")
12         out.println("<br>");
13
14      %>
15     </body>
16   </html>
```

【说明】

(1) 没写语句结束符分号的错误。

在本例程序的第 11 行语句没写分号，浏览器访问该页面时，会显示如图 4-19 所示的错误信息。



HTTP Status 500 -

type Exception report

message

description The server encountered an internal error () that prevented it from fulfilling this request.

exception

```
org.apache.jasper.JasperException: Unable to compile class for JSP:

An error occurred at line: 11 in the jsp file: /multi99.jsp
Syntax error, insert ";" to complete Statement
8: <%
9: for(int i=1;i<=9;i++){
10:    for(int j=1;j<=i;j++)
11:       out.print(j+"*"+i+"="+i*j+" ")
12:       out.println("<br>");
13:     }
14: %>
```

图 4-19　没写分号的错误信息

从图中可以看到，这个错误可通过页面的错误信息定位错误的位置并改正。

(2) 变量没有定义就使用。

在本例程序的第 10 行变量 j 没有定义就使用，在发生这样的错误时，浏览器访问该页面会显示如图 4-20 所示的错误信息。

根据浏览器中显示的错误信息 j cannot be resolved，可以判断出变量 j 没有被定义。

(3) 多写或少写花括号。

在多重循环语句或嵌套的条件语句中，有时会由于少写或多写花括号而引起错误，如本例的第 13 行就少写了一个花括号。浏览器访问该页面时，会显示如图 4-21 所示的错误信息。

根据浏览器中显示的错误信息，可以判断出少写花括号而引起的错误，并逐一核对花括号的对应关系，找出错误。

```
HTTP Status 500 -

type Exception report

message

description The server encountered an internal error () that prevented it from fulfilling this request.

exception

org.apache.jasper.JasperException: Unable to compile class for JSP:

An error occurred at line: 10 in the jsp file: /multi99.jsp
j cannot be resolved
7: <font color = "blue" >
8: <%
9: for(int i=1;i<=9;i++){
10:    for(j=1;j<=i;j++)
11:       out.print(j+"*"+i+"="+i*j+" ");
12:       out.println("<br>");
13:     }
```

图 4-20 变量 j 没有被定义的错误信息

```
HTTP Status 500 -

type Exception report

message

description The server encountered an internal error () that prevented it from fulfilling this request.

exception

org.apache.jasper.JasperException: Unable to compile class for JSP:

An error occurred at line: 70 in the generated java file
Syntax error, insert "}" to complete Block

An error occurred at line: 80 in the generated java file
Syntax error on token "}", delete this token

An error occurred at line: 81 in the generated java file
Syntax error, insert "}" to complete ClassBody
```

图 4-21 少写花括号的错误信息

改正所有错误后,本程序正确的输出结果如图 4-22 所示。

```
1*1=1
1*2=2 2*2=4
1*3=3 2*3=6 3*3=9
1*4=4 2*4=8 3*4=12 4*4=16
1*5=5 2*5=10 3*5=15 4*5=20 5*5=25
1*6=6 2*6=12 3*6=18 4*6=24 5*6=30 6*6=36
1*7=7 2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49
1*8=8 2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64
1*9=9 2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81
```

图 4-22 multi99.jsp 的正确输出

4.7.3 JSP 运行错误的调试

运行错误是当 JSP 引擎将 JSP 文件编译成的 Servlet 加载到内存执行时发生的错误,如数组越界、除零及一些数据操作都可能导致运行错误。本节以一个数组越界程序为例介绍这种错误的调试方法。

【例 4-14】 数组越界程序。

本例源代码如下：

```
                    <!-- 例程 4-20 array.jsp -->
<%@ page language="java" contentType="text/html;charset=gb2312" %>
<%
    int a[] = new int[10];
    for(int i = 0;i<20;i++)
        System.out.println(a[i] = i);
%>
```

浏览器访问该页面时显示如图 4-23 所示的错误信息。

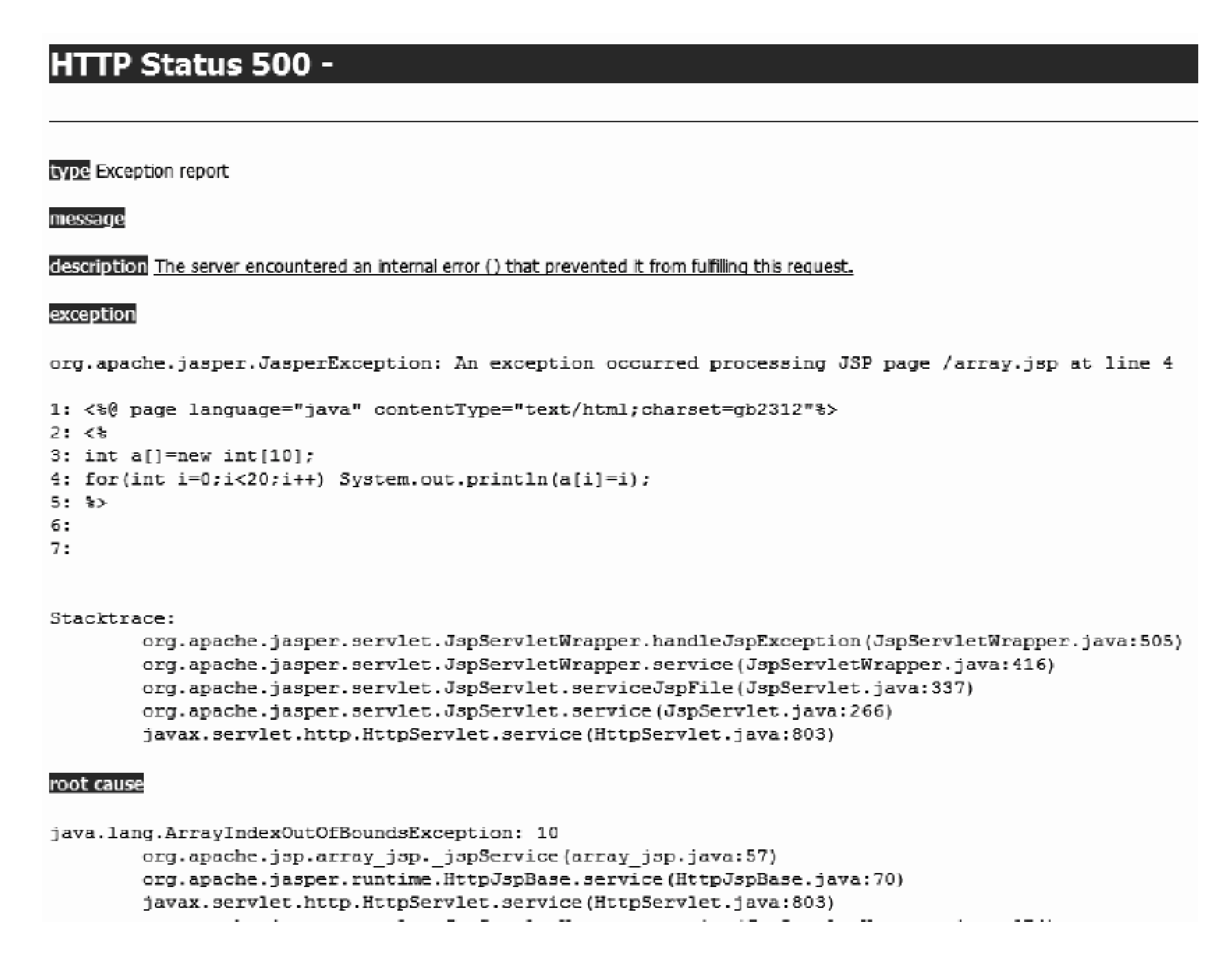


图 4-23 数组越界的错误信息

根据浏览器中显示的错误信息：

```
java.lang.ArrayIndexoutOfBoundsexception: 10
org.apache.jsp.array_jsp._jspService(array_jsp.java:57)
```

可以判断出错误发生在 array_jsp.java 的 57 行程序中，array_jsp.java 是 array.jsp 文件被编译成 Servlet 所对应的 Java 文件，它被存放在 Tomcat 目录下的 work 目录中。图 4-24 所示是 work 目录的结构，可以找到 array_jsp.java 文件。

array_jsp.java 的源代码如下：

```
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
public final class array_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent
{
```

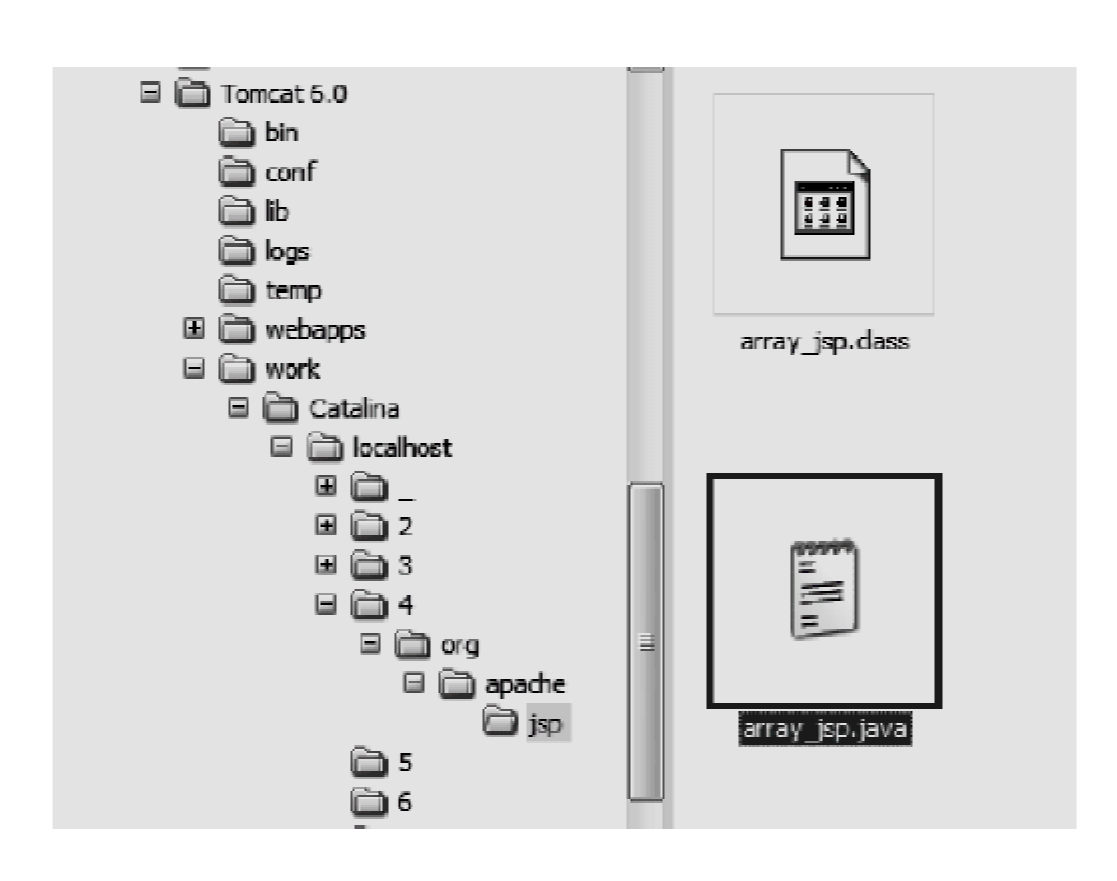


图 4-24　work 的目录结构

```
private static final JspFactory _jspxFactory = JspFactory.getDefaultFactory();
private static java.util.List _jspx_dependants;
private javax.el.ExpressionFactory _el_expressionfactory;
private org.apache.AnnotationProcessor _jsp_annotationprocessor;
public Object getDependants()
{
  return _jspx_dependants;
}
public void _jspInit()
{
   _el_expressionfactory = _jspxFactory.getJspapplicationContext(getServletconfig().
getServletContext()).getExpressionFactory();
    _jsp_annotationprocessor = (org.apache.AnnotationProcessor) getServletconfig().
getServletContext().getAttribute(org.apache.AnnotationProcessor.class.getName());
}
public void _jspDestroy()
{
}
public void _jspService(HttpServletRequest request, HttpServletResponse response)
      throws java.io.IOexception, Servletexception
{
  PageContext pageContext = null;
  Httpsession session = null;
  ServletContext application = null;
  Servletconfig config = null;
  JspWriter out = null;
  Object page = this;
  JspWriter _jspx_out = null;
  PageContext _jspx_page_context = null;
  try
  {
    response.setContentType("text/html;charset=gb2312");
    pageContext = _jspxFactory.getPageContext(this, request, response,
              null, true, 8192, true);
    _jspx_page_context = pageContext;
    application = pageContext.getServletContext();
```

```
        config = pageContext.getServletconfig();
        session = pageContext.getsession();
        out = pageContext.getout();
        _jspx_out = out;
        out.write('\r');
        out.write('\n');
        int a[] = new int[10];
        for(int i = 0;i < 20;i++ ) System.out.println(a[i] = i);
        out.write('\r');
        out.write('\n');
        } catch (Throwable t)
        {
         if (!(t instanceof SkipPageexception)){
           out = _jspx_out;
           if (out != null && out.getBufferSize() != 0)
             try
             {
             out.clearBuffer();
             } catch (java.io.IOexception e) {}
           if (_jspx_page_context != null) _jspx_page_context.handlePageexception(t);
        }
      } finally {
        _jspxFactory.releasePageContext(_jspx_page_context);
      }
    }
  }
```

从这个 Java 程序的 for(int i=0;i<20;i++) System.out.println(a[i]=i);语句中就可以定位错误并改正 JSP 程序中对应的代码了。

4.8 上机指导与练习

4.8.1 用户注册

1. 练习目标

(1) 熟悉表单的使用。

(2) 了解散列表对象的使用(hashtable)。

2. 练习指导

本程序由两个 JSP 页面文件构成。由一个页面 Zhuce.jsp 创建一个接受用户名输入的窗口,将用户名提交给 login.jsp 页面,login.jsp 获取用户名,实现用户注册。

(1) 建立 Zhuce.jsp 页面,创建一表单,其中包含一个文本框,客户在此文本框中输入用户名。

(2) 建立 login.jsp 页面,创建一空的散列对象(hashtable)。

(3) 定义一个 putName(String s)方法,该方法以 s 为属性名和属性值,插入到散列对

象(hashtable)中。

(4) 从表单中获取用户名,若散列对象中没有注册过此用户名,则将(name,name)插入到散列对象中(表示实现了用户注册),否则,提示用户换个名字注册。

该程序源代码如下:

```
<!-- 例程 4-21 zhuce.jsp -->

<%@ page contentType="text/html;charset=GB2312" %>
<html>
<body>
<font size=3>
  <form action="login.jsp" method=post>
    <p>请输入你的姓名:
    <INPUT  TYPE=TEXT     NAME="name"   value="abc">   <br>
    <input  type="submit"   name=submit   value="注册">
  </form>
</font>
</body>
</html>
```

```
<!-- 例程 4-22 login.jsp -->
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.util.*" %>
<html>
<body  bgcolor=cyan>
<font size=3>
    <%!
      Hashtable hashtable=new Hashtable();
      public  synchronized void putName(String s)
      {
        hashtable.put(s,s);
      }
     %>
    <%
      String name=request.getParameter("name");
      if(name==null)   name=" ";
      byte b[]=name.getBytes("ISO-8859-1");
      name=new String(b);
      if(!(hashtable.containsKey(name)))
          {
            putName(name);
            out.print("<br>"+"你已注册成功");
            out.print("<br>"+"你注册的名字是"+name);
          }
      else
        out.print("<br>"+"该名字已存在,请换个名字");
     %>
</font>
</body>
</html>
```

本程序源代码存放于本书配套素材中的 zhuce.jsp 文件及 login.jsp 文件中(文件路径为 code\4\上机指导)。

4.8.2 信息的保存和获取

1. 练习目标

(1) 熟练使用 session 对象保存会话信息。
(2) 熟悉 session 对象中的信息共享。

2. 练习指导

(1) 本程序的功能是将客户的姓名和通信地址保存在 session 对象中,实现同一个 Web 目录下的页面对 session 对象中的信息共享。

(2) 程序创建 3 个页面,第一个页面用来输入姓名,第二个页面输入通信地址,第三个页面实现信息获取,3 个页面文件保存在同一个 Web 目录中。

(3) 在第一个页面文件 name.jsp 中,创建一个包含文本控件的表单,用此文本控件输入姓名。

(4) 在第二个页面文件 address.jsp 中,获取客户端输入的姓名(xm),将其添加到 session 对象中,再创建一个包含文本控件的表单,用此文本控件输入通信地址。

(5) 在第三个页面文件 account.jsp 中,获取客户端输入的通信地址(dz),将其添加到 session 对象中。

(6) 从 session 对象中获取姓名和通信地址。

(7) 最后将姓名和通信地址输出到客户端。

本程序源代码如下:

```
<!-- 例程 4-23 name.jsp -->

<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY  bgcolor=cyan><FONT size=3>
 <P>请输入您的姓名:
  <FORM action="address.jsp" method=post name=form>
     <INPUT type="text" name="buy_name">
     <INPUT TYPE="submit" value="提交姓名" name=submit>
  </FORM>
</BODY>
</HTML>
```

```
<!-- 例程 4-24 address.jsp -->

<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY  bgcolor=cyan><FONT size=3>
  <%
    String xm=request.getParameter("buy_name");
```

```
     session.setAttribute("name",xm);
   %>
  <P>请输入您的通信地址：
  <FORM action = "account.jsp" method = post name = form >
      < INPUT type = "text" name = "shangpin">
      < INPUT TYPE = "submit" value = "提交地址" name = submit >
  </FORM >
</BODY >
</HTML >
```

```
<!-- 例程 4 - 25 account.jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<HTML>
<BODY bgcolor = cyan><FONT Size = 3 >
   <%!  //处理字符串的方法
      public String getString(String s)
      {
         if(s == null)
           {
             s = "";
           }
         try{
              byte b[ ] = s.getBytes("ISO - 8859 - 1");
              s = new String(b);
           }
         catch(exception e)
           {
           }
         return s;
     }
   %>
   <%
      String sp = request.getParameter("shangpin");
      session.setAttribute("goods",sp);
   %>
  <%
    String xinming = (String)session.getAttribute("name");
    String shangpin = (String)session.getAttribute("goods");
    xinming = getString(xinming); //对姓名进行编码
    shangpin = getString(shangpin);//对通信地址进行编码
   %>
   <P>您好!
   <P>您输入的姓名是：<% = xinming %>
   <P>您输入的通信地址是：<% = shangpin %>
</BODY >
</HTML >
```

本程序源代码存放于本书配套素材的相应文件中(文件路径为 code\4\上机指导)。

4.8.3 猜数字游戏

1. 练习目标

(1) JSP 基本语法的综合运用。
(2) JSP 内置对象的综合运用。

2. 练习指导

该程序由 5 个页面组成。

(1) index.jsp 页面：猜数字游戏的主页，系统随机生成一个 1～100 之间的数，要求在文本框中输入要猜的数。

(2) process.jsp 页面：判断所猜的数和系统生成的数之间的关系。

(3) smaller.jsp 页面：提示猜的数小于生成的数，并要求重新输入。

(4) larger.jsp 页面：提示猜的数大于生成的数，并要求重新输入。

(5) ok.jsp 页面：提示已经猜中。

猜数字游戏程序的源代码如下：

```
<!-- 例程 4-26 index.jsp -->
```

```
<%@ page contentType="text/html;charset=gb2312" %>
<html>
<head>
<title>猜数游戏</title>
</head>
<body>
 <p align="center"><font color="#0000FF" size="4">猜数游戏</font>
 <%
     int num=(int)(Math.random()*100)+1;
     session.setAttribute("count",new Integer(0));
     session.setAttribute("num",new Integer(num));
  %>
  <p><font size="3" color="#0000FF">系统随机生成一个 1～100 之间的数</font>
<P><font size="3" color="#0000FF">请在下面的文本框中输入你猜的数</font>
<form action="process.jsp" method="post" name="form1">
  <input type="text" name="number" >
  <input type="submit" value="提交">
</form>
</body>
</html>
```

```
<!-- 例程 4-27 process.jsp -->
```

```
<%@ page contentType="text/html;charset=gb2312" %>
<html>
<head>
<title>猜数游戏-处理</title>
```

```
</head>
<%
    String num = request.getParameter("number");
     if(num == null)
       {
           num = "0";
       }
      int gNum = Integer.parseInt(num);
      Integer i = (Integer)session.getAttribute("num");
      int rNum = i.intValue();
     if(gNum == rNum)
        {
          int count = ((Integer)session.getAttribute("count")).intValue();
          count ++ ;
          session.setAttribute("count",new Integer(count));
          response.sendRedirect("ok.jsp");
        }
       else if(gNum > rNum)
        {
          int count = ((Integer)session.getAttribute("count")).intValue();
          count ++ ;
          session.setAttribute("count",new Integer(count));
          response.sendRedirect("larger.jsp");
        }
    else if(gNum < rNum)
        {
          int count = ((Integer)session.getAttribute("count")).intValue();
          count ++ ;
          session.setAttribute("count",new Integer(count));
          response.sendRedirect("smaller.jsp");
        }
 %>
</html>
```

<!-- 例程 4 - 28 smaller.jsp -->

```
<%@ page contentType = "text/html;charset = GB2312" %>
<html>
<head>
<title>猜数游戏 - 小</title>
</head>
<body>
<p><font color = "blue">
   你猜的数比系统生成的数小,请再猜一次,在下面的文本框中输入你猜的数:
</font>
 <form action = "process.jsp" method = "post" name = "form1">
   <input type = "text" name = "number" >
   <input type = "submit" value = "提交">
</form>
</body>
</html>
```

```
                    <!-- 例程 4 - 29 larger.jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<html>
<head>
<title>猜数游戏 - 大</title>
</head>
<body>
<p><font color = "blue">
你猜的数比系统生成的数大,请再猜一次,在下面的文本框中输入你猜的数:
</font>
 <form action = "process.jsp" method = "post" name = "form1">
  <input type = "text" name = "number" >
  <input type = "submit" value = "提交">
</form>
</body>
</html>
```

```
                    <!-- 例程 4 - 30ok.jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<html>
<head>
<title>猜数游戏 - 答对了</title>
</head>
<body>
<p><font color = "blue">恭喜你,答对了!</font>
 </body>
</html>
```

本程序源代码存放于本书配套素材的相应文件中(文件路径为 code\4\上机指导)。

本章小结

本章介绍了 JSP 内置对象的概念、生命周期、作用范围和对象方法的实际应用。使用 JSP 内置对象,可以方便地操作页面属性和行为,访问页面运行环境,实现页面内、页面间、页面与环境之间的通信和相互操作。另外,在本章的最后还介绍了 JSP 程序常用的调试方式。通过本章的学习,读者可以掌握 JSP 基本的编程方法。

习题 4

一、简答题

1. 简述 application 对象与 session 对象的不同。
2. 简述 request 对象的功能。
3. JSP 程序的错误分哪几类?

二、操作编程题

1. 设计一个 JSP 页面，要求页面的颜色每天都发生变化。
2. 编写 JSP 程序，在网页上输出下面的图形。

```
*
* *
* * *
* * * *
* * * * *
```

3. 改写猜数字游戏，要求最后能够显示出共猜了几次。

第5章 数据库操作

数据库是许多企业应用中不可缺少的环节，而且在大多数 Web 应用程序中都用到了数据库从而实现了信息的交互。理解和掌握如何在 Web 应用中使用数据库对 JSP 开发人员来说非常重要。本章将系统地介绍页面与数据库之间的通信。

本章主要内容：

- 数据库概述；
- JDBC 的结构；
- JDBC 的驱动程序；
- 数据库的连接与操作。

5.1 数据库概述

数据库，简单地说就是存储各种数据、信息的容器，根据需要，将页面数据保存到数据库中，或者将数据库中的数据展现在页面上。目前，大型 JSP 项目的开发都离不开数据库，如何设计开发一个数据结构合理、功能完善的数据库，对 JSP Web 开发是非常重要的，它可能直接影响到 Web 应用程序的运行效率和稳定性。

5.1.1 关系模型

数据库管理系统是管理数据库的系统，它按一定的数据模型组织数据。数据库管理系统采用的数据模型主要有：关系模型、层次模型和网状模型。关系模型是目前应用最广的数据模型，以关系模型管理数据库的管理系统很多，例如 Access、Visual FoxPro、SQL Server、Sybase、Oracle、DB2 等。

关系模型中数据的逻辑结构是一张二维表，它由行和列组成。例如，学生信息登记表，如表 5-1 所示。

表 5-1 学生信息登记表

学号	姓名	年龄	性别	系部	年级
95004	王小萌	19	女	社会学	95
95006	黄鹏	20	男	商品学	95
95008	张文斌	18	女	法律学	95

关系模型以二维表格(关系表)的形式组织数据库中的数据。在关系表中,表格中的一行称为一个记录,一列称为一个字段,每列的标题称为字段名。如果给每个关系表取一个名字,则有 n 个字段的关系表的结构可表示为:关系表名(字段名 1,……,字段名 n),通常把关系表的结构称为关系模式。关系模型的基本概念如下所述。

(1) 关系:一个关系对应一张表,如表 5-1 所示。

(2) 元组:表中的一行即为一个元组(记录)。

(3) 属性:表中的一列即为一个属性(字段),给每一个属性起一个名称即属性名。

(4) 主码:表中的某属性组,它可以唯一确定一个元组。

(5) 域:属性的取值范围,如年龄一般在 1～100 岁之间,性别的域是(男,女)。

(6) 分量:元组中的一个属性值(字段值)。

(7) 关系模式:对关系的描述。

(8) 格式:关系名(属性 1,属性 2,……,属性 n)。例如,学生(学号、姓名、年龄、性别、系、年级)。

5.1.2 结构化查询语言 SQL

结构化查询语言 SQL 是用于操作关系数据库的标准语言,目前,许多关系型数据库供应商都在自己的数据库中支持 SQL 语言,例如 Access、Oracle、Sybase、Infomix、DB2 和 Microsoft SQL Server 等。SQL 虽然名为查询语言,但实际上具有数据定义、查询、更新和控制等多种功能,它使用方便、功能丰富、简洁易学。

SQL 语言由 3 部分组成:

(1) 数据定义语言(DDL):用于执行数据库定义的任务,对数据库以及数据库中的各种对象进行创建、删除、修改等操作。数据库对象主要包括表、默认约束、规则、视图、触发器和存储过程。

(2) 数据操纵语言(DML):用于操纵数据库中各种对象,检索和修改数据。

(3) 数据控制语言(DCL):用于安全管理,确定哪些用户可以查看或修改数据库中的数据。

下面介绍 SQL 语言中最常用的命令。

1. 创建数据库 CREATE DATABASE

在 SQL 语言中,创建一个新数据库的基本语法格式如下:

```
CREATE DATABASE 数据库名称
```

数据库名称在服务器中必须唯一,并且符合标识符的命名规则。

2. 创建表 CREATE TABLE

在 SQL 语言中,创建一个新表的基本语法格式如下:

```
CREATE TABLE 表名称 (列名 数据类型, …)
```

表的名称必须符合标识符命名规则,列名又称字段名,必须符合标识符规则,并且在表内唯一。数据类型可以是系统数据类型或用户定义数据类型。

3. 插入数据语句 INSERT

INSERT 可添加记录到表中，其语法形式如下：

```
INSERT INTO 表名 [(字段名表)] VALUES (值表)
```

例如，向 XS 表添加一条记录，并给所有字段赋值：

```
INSERT INTO XS  VALUES('051216','罗林琳', 0 ,'1985-30-1', '计算机', 40,NULL)
```

4. 删除数据语句 DELETE

DELETE 用来从表中删除记录，其语法格式如下：

```
DELETE FROM 表名 [WHERE 条件]
```

例如，从 XS 表中删除姓名为“林时”的记录：

```
DELETE FROM XS   WHERE XM = '林时'
```

5. 更新数据语句 UPDATE

UPDATE 语句用来更新表中的记录，其语法格式如下：

```
UPDATE 表名 SET 字段名 1 = 值 [,字段名 2 = 值 …][WHERE 条件]
```

例如，将计算机系的学生的总分增加 2：

```
UPDATE XS SET ZXF = ZXF + 2  WHERE ZY = '计算机'
```

6. 数据查询 SELECT

数据查询是非常常见的数据库操作，数据查询使用 SELECT 语句，其语法形式是：

```
SELECT [DISTINCT] [别名.]字段名或表达式 [AS 列标题]
FROM 表或视图 别名
[ WHERE 条件]
[ GROUP BY 分组表达式]
[ ORDER BY 排序表达式[ ASC | DESC ]]
```

其中 SELECT 子句指出查询结果中显示的字段名或字段名和函数组成的表达式等，AS 列标题指定查询结果显示的列标题。若要显示表中所有字段时，可用通配符“*”代替字段名列表。可用 DISTINCT 去除重复的记录行。

- FROM 子句指定表或视图。
- WHERE 子句定义了查询条件。
- GROUP BY 子句对查询结果分组。
- ORDER BY 子句对查询结果排序。

SELECT 子句虽然复杂，但在实际应用中，几乎不可能同时遇到这么多选项，一般常用的形式是：

```
SELECT  [别名.]字段名或表达式
FROM 表或视图 别名
[WHERE 条件]
```

例如：

(1) 查询 XSCJ 数据库的 XS 表中各个同学的姓名和总学分。

```
SELECT XM, ZXF FROM XS
```

(2) 查询 XS 表中各个同学的所有信息。

```
SELECT * FROM XS
```

(3) 查询 XS 表中总学分大于等于 50 的同学的情况。

```
SELECT * FROM XS WHERE ZXF >= 50
```

5.2 JDBC 技术

与数据库的交互是动态网站一个重要的组成部分，在 JSP 中使用 JDBC 技术来实现与数据库的连接。JSP 提供了 JSP 操作数据库的各种接口程序，所以 JDBC 数据库编程对 Web 开发是非常重要的。

5.2.1 JDBC 介绍

前面介绍了数据库知识和 SQL 语言，在设计数据库应用程序时，首先需要连接到要访问的数据库，解决此问题的方法是采用数据库接口技术。目前在市面上最常用的两种数据库接口是开放数据库连接(Open Database Connectivity，ODBC)和 Java 数据库连接(Java Database Connectivity，JDBC)。

JDBC 是一种可用于执行 SQL 语句的 Java API(应用程序设计接口)，它由一些 Java 语言编写的类和界面组成。JDBC 为数据库应用开发人员、数据库前台工具开发人员提供了一种标准的应用程序设计接口，使开发人员可以用纯 Java 语言编写完整的数据库应用程序。

通过使用 JDBC，可以很方便地将 SQL 语句传送给几乎任何一种数据库，也就是说，开发人员可以不必写一个程序访问 Oracle，再写一个程序访问 SQL Server。不但如此，使用 Java 编写的应用程序可以在任何支持 Java 的平台上运行，不必在不同的平台上编写不同的应用程序。Java 和 JDBC 的结合可以在开发数据库应用时真正实现“一次开发，随处运行。”

简单地说，JDBC 能实现以下 3 个功能：

(1) 同一个数据库建立连接。

(2) 向数据库发送 SQL 语句。

(3) 处理数据库返回的结果。

5.2.2 JDBC 体系结构

在 JDK 1.1 版本之前，Java 语言提供的对数据库访问支持的能力是很弱的，编程人员不得不使用 ODBC 函数调用，这使得 Java 程序的跨平台发布能力受到很大的限制。JDBC

的出现使 Java 程序对各种数据库的访问能力大大增强。JDBC 的体系结构如图 5-1 所示。

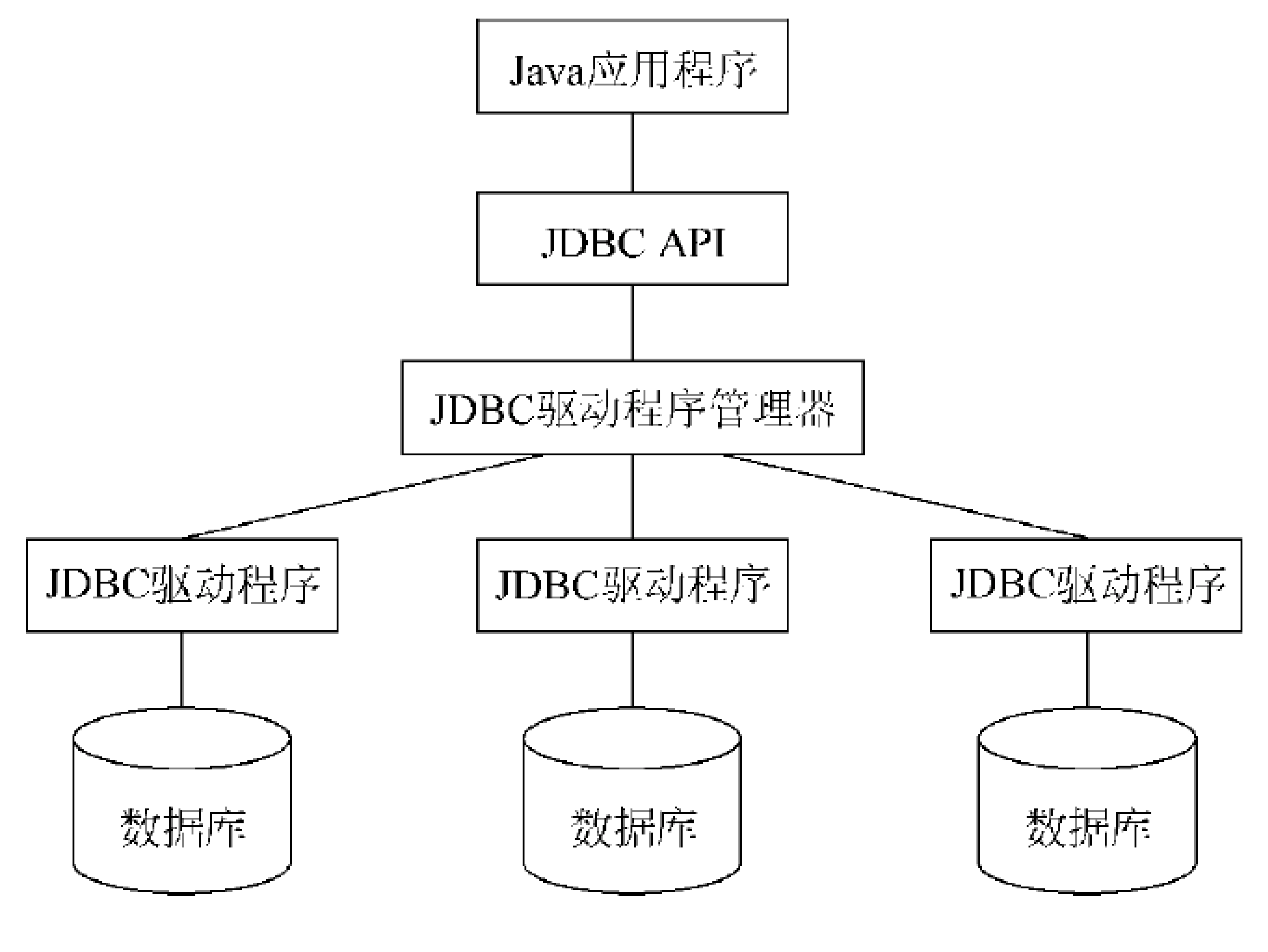


图 5-1 JDBC 结构图

由图 5-1 中可以看出，JDBC 的体系结构有 4 个组件，分别为应用程序、JDBC API、JDBC 驱动程序管理器和为各种数据库定制的 JDBC 驱动程序，提供与不同数据库的透明连接。其中 JDBC API 的作用就是屏蔽不同的数据库间 JDBC 驱动程序之间的差别，使得程序设计人员有一个标准的、纯 Java 的数据库程序设计接口，为在 Java 中访问任意类型的数据库提供技术支持。JDBC 驱动程序管理器为应用程序装载数据库驱动程序。JDBC 驱动程序与具体的数据库相关，用于建立与数据源的连接，以及向数据库提交 SQL 请求。

5.2.3 JDBC 驱动程序

JDBC 驱动程序按其实现方式的不同可以分为 4 种类型，不同类型的驱动程序有着不一样的使用方法，所以在连接数据库之前，必须选择一种适当的驱动程序。

1. JDBC-ODBC 桥

在 JDBC 刚刚产生时，JDBC-ODBC 桥是很有用的。通过 JDBC-ODBC 桥，开发者可以使用 JDBC 来访问一个 ODBC 数据源。JDBC-ODBC 桥驱动程序为 Java 应用程序提供了一种把 JDBC 调用映射为 ODBC 调用的方法。因此，需要在客户端机器上安装 ODBC 驱动程序。这种类型的驱动程序最适合于企业网或是用 Java 编写的三层结构的应用程序服务器代码。

该驱动程序的工作原理如图 5-2 所示。

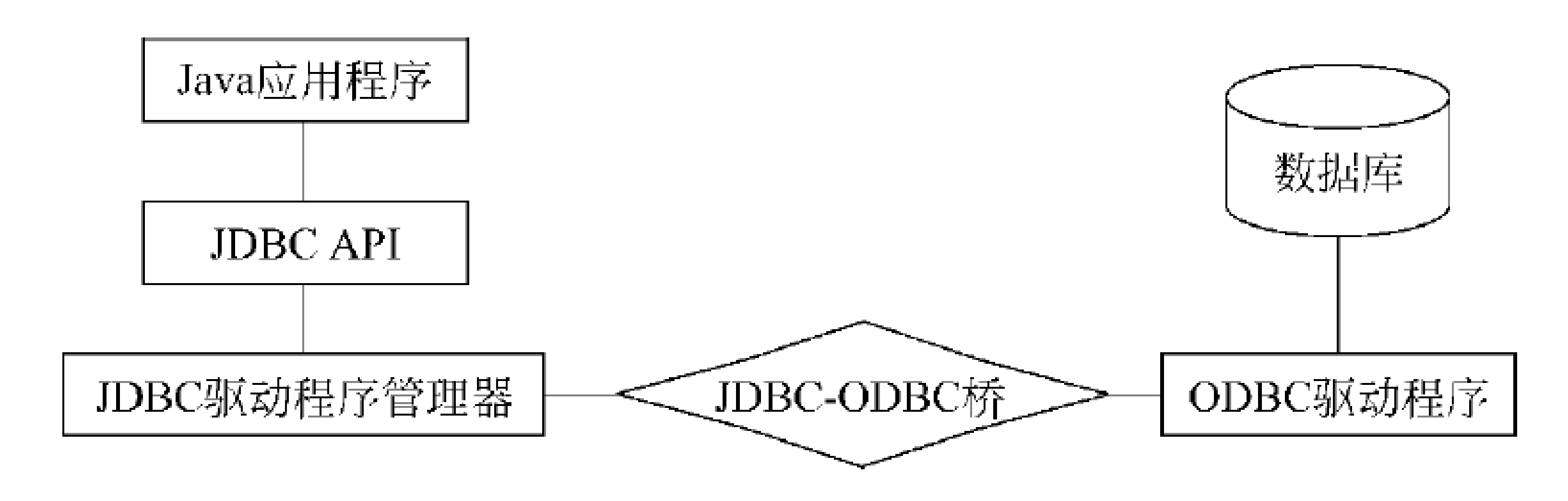


图 5-2 JDBC-ODBC 桥的工作原理

2．Java 到本地 API

该类型的驱动程序把客户机 API 上的 JDBC 调用转换为其他数据库管理系统的调用。这也是一种桥驱动程序，它使用 Java 实现与数据库厂商专有的 API 的混合形式来提供数据访问。JDBC 驱动将标准的 JDBC 调用转变为对数据库 API 的本地调用。与第一种类型类似，使用这种类型的驱动程序也需要在客户端机器上安装厂商专有的 API。

该驱动程序的工作原理如图 5-3 所示。

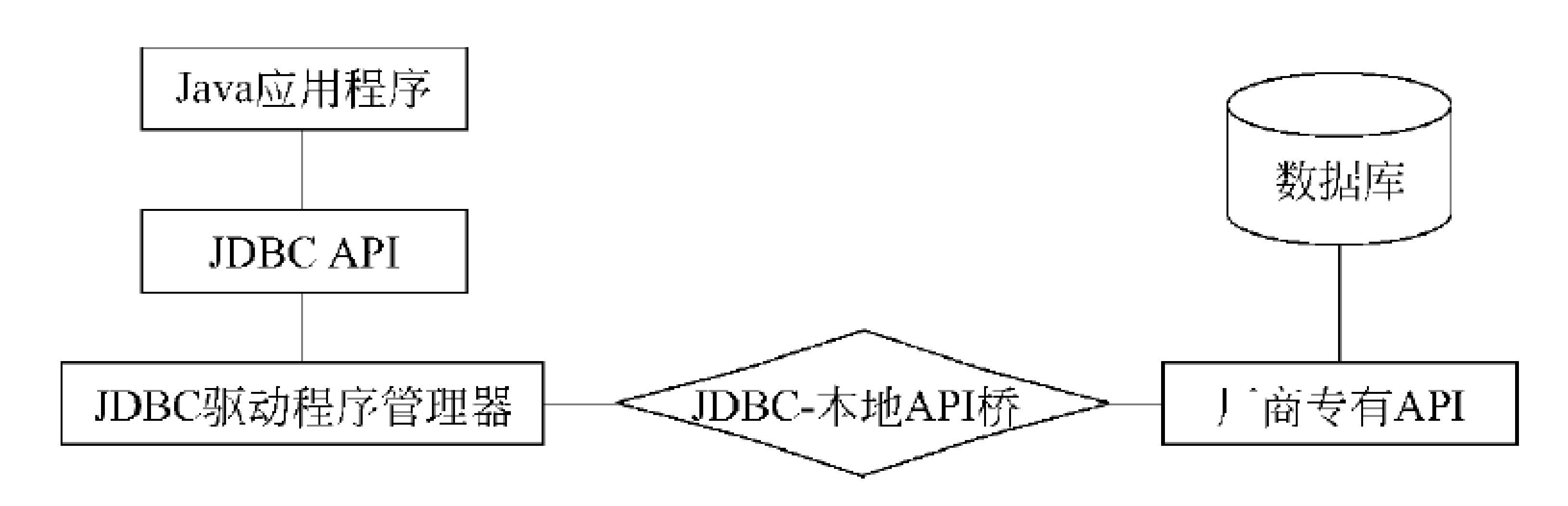


图 5-3　Java 到本地 API 的工作原理

3．JDBC 网络纯 Java 驱动程序

这种类型的驱动程序将 JDBC 转换为与数据库管理系统无关的网络协议，之后这种协议又被某个服务器转换为一种数据库管理系统协议。这种网络服务器中间件能够将它的纯 Java 客户机连接到多种不同的数据库上，所用的具体协议取决于提供者。

通常，这是最为灵活也是最成熟的 JDBC 驱动程序，不但无须在客户端机器上安装任何额外的驱动程序，也不需要在服务器端安装任何中间程序，所有存取数据库的操作都直接由驱动程序来完成。

该驱动程序的工作原理如图 5-4 所示。

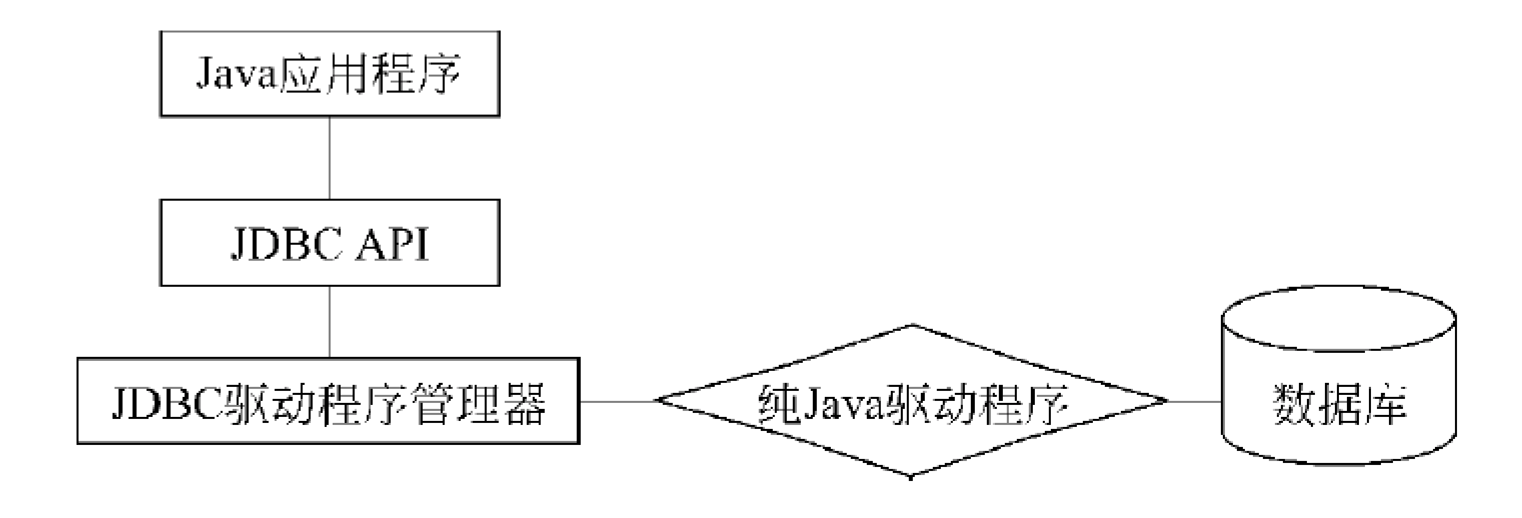


图 5-4　JDBC 网络纯 Java 驱动程序的工作原理

4．Java 到本地数据库协议

该类型的驱动程序将 JDBC 调用直接转换为 DBMS 所使用的网络协议。这种方式使用一个中间数据库访问服务器，通过这个服务器，可以把 Java 客户端连接到多个数据库服务器上。该类型的驱动程序最大的好处是省去了在客户端安装任何驱动程序的麻烦，只要在服务器端安装好数据访问中间服务器，中间服务器会负责所有存取数据库时必要的转换。

该驱动程序的工作原理如图 5-5 所示。

对于以上 4 类驱动程序的选择，需要考虑构建应用程序的实际需要。一般建议不使用桥驱动程序，即第 1、2 类驱动程序，它们主要是作为纯 Java 驱动程序还没有上市之前的过

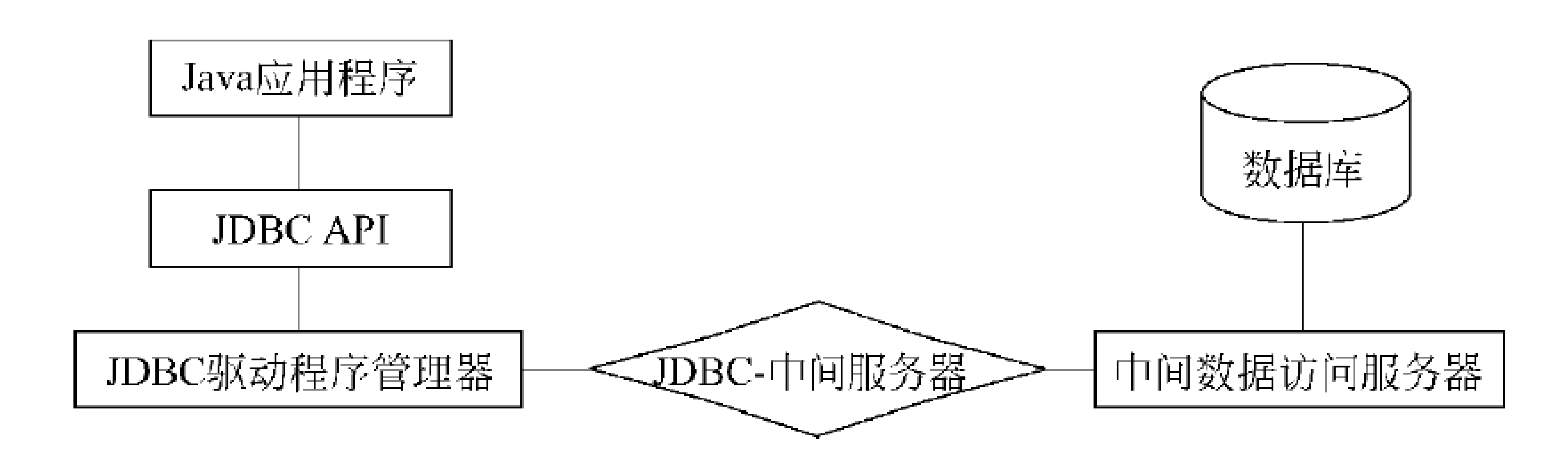


图 5-5 Java 到本地数据库协议的工作原理

渡方案来使用,效率相对较低,程序的可移植性差。而第 3、4 类驱动程序是从 JDBC 访问数据库的首选方法,它们不但使程序的可移植性提高,达到跨平台的目的,还省去了在客户端安装驱动程序的麻烦。

5.2.4 JDBC 接口

JDBC 的接口分为两个层次:一个是面向程序开发人员的 JDBC API;另一个是底层的 JDBC Driver API。

1. 面向程序开发人员的 JDBC API

JDBC API 被描述成为一组抽象的 Java 接口,使得应用程序可以对某个数据库打开连接,执行 SQL 语句并处理结果。

JDBC API 主要包括以下接口。

1) java.sql.DriverManager

用于处理驱动的调入并对新产生的数据库连接提供支持。DriverManager(驱动程序管理器)负责管理 JDBC 驱动程序,在使用 JDBC 驱动程序之前,必须先将驱动程序加载并向 DriverManager 注册后才可使用。

2) java.sql.Connection

Connection 对象通过 DriverManager.getConnection()方法获得,代表与特定的数据库的连接,也就是在已经加载的驱动程序和数据库之间建立连接。Connection 接口是 JSP 编程中使用最频繁的接口之一。

3) java.sql.Statement

Statement 用来执行静态 SQL 语句,该接口代表一个特定的容器,用来对一个特定的数据库执行 SQL 语句。该接口类型又有两个子类型。

- java.sql.PreparedStatement:用于执行预编译的 SQL 语句。
- java.sql.CallableStatement:用于执行对一个数据库内嵌过程的调用。

4) java.sql.ResultSet

本接口控制对一个特定语句的行数据的存取。在 Statement 执行 SQL 查询语句时,会返回 ResultSet 查询结果记录集,ResultSet 接口提供了逐行访问这些记录的方法。

2. JDBC Driver API

JDBC Driver API 是面向驱动程序开发商的编程接口。对于大多数数据驱动程序来

说，仅实现 JDBC API 提供的抽象类就可以了。也就是说，每个驱动程序必须提供对于 java. sql. Connection、java. sql. Statement、java. sql. PreparedStatement 和 java. sql. ResultSet 等主要接口的实现方法。如果目标 DBMS 提供有 OUT 参数的内嵌过程，那么还必须提供 java. sql. CallableStatement 接口。当 java. sql. DriverManager 需要为一个特定的数据库 URL 加载驱动程序时，每个驱动程序就需要提供一个能实现 java. sql. Driver 接口的类。

5.3 连接数据库

与数据库的交互是动态 Web 应用程序的一个重要的组成部分。JSP 使用 JDBC 技术来实现与数据库的连接，它提供了 JSP 操作数据库的各种接口程序。很多数据库系统都具有 JDBC 驱动程序，JSP 可以直接利用它来访问数据库。使用 JDBC 连接数据库存取数据时，必须执行以下 3 个步骤。

（1）用 DriverManager 加载及注册适当的 JDBC 驱动程序。

（2）用 JDBC URL 定义驱动程序与数据源之间的连接，并且建立一个连接对象。

（3）建立一个 SQL 陈述式对象（Statement Object），并且利用它来执行 SQL 语句。

这 3 个步骤是 JDBC 连接数据库时最基本最必要的步骤，不管使用哪种数据库，都要经过这 3 个步骤。本节主要介绍 JSP 如何实现与 SQL Server 2000 数据库及 Access 数据库的连接。

5.3.1 JDBC 连接 SQL Server 数据库

SQL Server 2000 数据库是目前应用最为广泛的数据库之一，其操作方便、功能强大的优势使其得到了广大用户的青睐。在 JSP 中，用户可以使用 JDBC-ODBC 桥驱动程序连接 SQL Server 数据库，也可以使用 JDBC 的驱动程序 Microsoft SQL Server 2000 Driver for JDBC 来直接连接。这里以第二种方法为例介绍 JSP 与 SQL Server 数据库的连接。

1. 下载并安装 JDBC 驱动程序

首先用户需要安装 Microsoft 公司 SQL Server 2000 Driver for JDBC 驱动程序，微软推出的 JDBC 驱动程序 SQL Server 2000 Driver for JDBC 可以实现直接与 SQL Server 数据库的连接，用户可以到微软的官方网站 http://www. microsoft. com/downloads 下载免费的 SQL Server 2000 Driver for JDBC Service Pack 4 的安装文件。下载完毕并安装后，将 \Microsoft SQL Server 2000 Driver for JDBC\lib 目录下的 msbase. jar、msutil. jar 和 mssqlserver. jar 这 3 个文件复制到 \Tomcat 6. 0\lib 下，并且要重新配置系统变量 classpath，然后把这 3 个. jar 文件的存放路径添加到 classpath 的值里面。

2. 加载驱动程序

在 JDBC 中，通常有两种加载驱动程序的方式。一种是将驱动程序加到 java. lang. System 的属性 jdbc. drivers 中，这是一个由 DriverManager 类加载的驱动程序类名的列表，用冒号分隔。在 JDBC 中的 java. sql. DriverManager 类初始化时，在 JVM 的系统属性中搜

索 jdbc. drivers 字段的内容。如果存在以冒号分隔的驱动程序名称，则 DriverManager 类加载相应的驱动程序。另一种方式是在程序中利用 Class. forName()方法加载指定的驱动程序，这样将显式地加载驱动程序。这种方式与外部设置无关，因此推荐使用这种加载驱动程序的方式。代码如下：

```
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
```

其中 com. microsoft. jdbc. sqlserver. SQLServerDriver 为 SQL Server 的 JDBC 驱动程序的别名。

3. 创建指定数据库的 URL

要建立与数据库的连接，首先要创建指定数据库的 URL。数据库 URL 对象与网络资源的统一资源定位类似，具体语法为如下：

```
jdbc:subProtocol:subName://hostname:port;DatabaseName = dbname
```

其中 jdbc 表示当前通过 Java 的数据库连接进行数据库的访问；subProtocol 表示通过某种驱动程序支持的数据库连接机制；subName 表示在当前连接机制下所采用驱动的具体名称；hostname 表示主机名；port 表示相应的连接端口；DatabaseName 是要连接的数据库的名称。

4. 建立与数据库的连接

创建了数据源，加载了驱动程序，应用程序还是不能连接到数据库。应用程序要访问数据库，还必须创建一个到数据库的连接，即创建一个连接对象。

连接通常是通过数据库的 URL 对象，利用 DriverManager 的 getConnection()方法建立的。创建数据库连接时，需要提供数据库的 URL 和驱动类型，并且提供访问数据库的用户名和密码。如果有多个 JDBC 驱动程序可以与给定的 URL 连接，DriverManager 将轮流在每个驱动程序上调用 Driver. connect()方法，并将用户传递给 DriverManager. getConnection()方法的 URL 传递给它们，对这些驱动程序进行测试，然后连接第一个可以成功连接到给定 URL 的驱动程序。

5. 访问数据库

连接到数据库后就可以访问数据库。首先需要使用 Connection 类对象的 createStatement()方法从指定的数据库连接，得到一个 Statement(java. lang 包)的实例 stmt，然后使用这个实例的 executeQuery()方法来执行 SQL 语句。其代码如下：

```
stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
```

java. sql. ResultSet 类用来保存 SQL 语句的执行结果，还可以存取结果中的数据。通常对数据库查询或修改等操作将返回一个包含查询结果 ResultSe 对象。例如，将从 authors 数据表中查询的结果保存在 ResultSet 对象的 rs 中，代码如下：

```
ResultSet  rs = stmt.executeQuery(sql);
```

其中字符串变量 sql 是 JSP 页面要执行的 SQL 语句，通过 sql 变量得到相应的 ResultSet 对象。

6. 关闭数据库连接，释放资源

对数据库访问结束后，应及时地关闭 ResultSet 对象、Statement 对象和 Connection 对象，释放所占的资源。释放这些资源需要使用 close()方法，实现代码如下：

```
rs.close();             //关闭 ResultSet 对象
stmt.close();           //关闭 Statement 对象
conn.close();           //关闭 Connection 对象
```

【例 5-1】 在 JSP 中连接 SQL Server 2000 数据库。

本例使用了 SQL Server 2000 中自带的 Pubs 数据库，创建一个 JSP 文件 sqlser.jsp，用于访问 authors 表中的数据，该程序代码如下。本例源代码存放于本书配套素材中的 sqlser.jsp 文件中(文件路径为 code\5)。

```
<!-- 例程 5-1 sqlser.jsp -->
<%@ page contentType="text/html; charset=gb2312" language="java" import="java.sql.*"
errorPage="" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>使用 JDBC 直接连接 SQL Server</title>
<style type="text/css">
</style>
</head>
<body>
<div align="center"><span class="style1">访问 SQL Server 数据库</span><BR>
</div>
<hr>
<BR>
<table border=2  align="center">
  <tr>
    <td>au_id</td>
    <td>au_lname</td>
    <td>au_fname</td>
    <td>phone</td>
    <td>address</td>
    <td>city</td>
    <td>state</td>
    <td>zip</td>
    <td>contract</td>
  </tr>
<% Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
//加载驱动程序
String  url="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=pubs";
//数据库连接串,pubs 为数据库的名称
```

```
String  user = "sa";                    //登录用户名
String  password = "";                  //登录密码
Connection  conn =  DriverManager.getConnection(url,user,password);
//创建数据库连接
Statement  stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_
UPDATABLE);
String  sql = "select  top 10 *  from  authors";   //取出 authors 表中的所有字段
ResultSet  rs = stmt.executeQuery(sql);            //执行 SQL 语句
while(rs.next())  {%>
   <tr>
    <td><% = rs.getString("au_id") %></td>
    <td><% = rs.getString("au_lname") %></td>
    <td><% = rs.getString("au_fname") %></td>
    <td><% = rs.getString("phone") %></td>
    <td><% = rs.getString("address") %></td>
    <td><% = rs.getString("city") %></td>
    <td><% = rs.getString("state") %></td>
    <td><% = rs.getString("zip") %></td>
    <td><% = rs.getString("contract") %></td>
    </tr>
<% } %>
<%  out.print("数据库操作成功"); %>
<%
      rs.close();            //关闭 ResultSet 对象
      stmt.close();          //关闭 Statement 对象
      conn.close();          //关闭 Connection 对象
 %>
</table>
</body>
</html>
```

在上面的这段代码中，使用 Class.forName()方法加载 SQL Server 的驱动程序，然后再创建要连接数据库的 URL，使用 DriverManager.getConnection()方法来创建数据库连接，这个方法有 3 个参数，即 url、user 和 password，分别用来指定要连接的数据库的 URL 地址、登录的用户名和密码。该程序运行后的效果如图 5-6 所示。

连接SQL Server数据库

数据库操作成功

au_id	au_lname	au_fname	phone	address	city	state
172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA
213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA
238-95-7766	Carson	Cheryl	415 548-7723	589 Darwin Ln.	Berkeley	CA
267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland Av. #14	San Jose	CA
274-80-9391	Straight	Dean	415 834-2919	5420 College Av.	Oakland	CA

图 5-6　sqlser.jsp 的运行效果

5.3.2　JDBC-ODBC 连接 Access 数据库

由于目前 JDBC 还不能实现对所有数据库的直接访问，因为不是所有的数据库提供商都提供 JDBC 驱动程序，例如 Access，所以 JSP 访问 Access 就只能通过 JDBC-ODBC 桥，使用 ODBC 驱动程序实现对数据库的访问。

1. 加载驱动程序

在 JDBC 连接到 ODBC 数据库之前，必须加载 JDBC-ODBC 桥的驱动程序，代码如下。

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver")
```

该语句使用了 Class 类（java.lang 包）中的 forName()方法载入该驱动程序的类 sun.jdbc.odbc.JdbcOdbcDriver，从而创建了该驱动程序的一个实例。

2. 创建数据库连接

加载 JDBC-ODBC 桥的驱动程序后，就可以连接数据库了。首先创建 Connection（java.lang 包）类的一个实例 conn，并使用 DriverManager()方法的 getConnection 来测试使用 url 指定的数据库连接。创建数据库连接的代码如下。

```
String url = "jdbc:odbc:dataname";
String user = "";
String password = "";
conn = DriverManager.getConnection(url,uesr,password);
```

这里的 java.sql.Connection 类用来管理 JDBC 和数据库之间的连接，还提供了对 SQL 语言的支持，以便操纵数据库、进行事务处理。java.sql.DriverManager 是驱动程序管理器，其参数 url 用来指定连接的 ODBC 数据源，user 和 password 用来指定访问数据库的用户名和密码。

3. 访问数据库

使用 Connection 类对象的 createStatement()方法从指定的数据库连接得到一个 Statement 的实例 stmt，然后使用这个实例的 executeQuery()方法来执行 SQL 语句，并将查询结果保存到 ResultSet 对象 rs 中。其代码如下：

```
stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_READ_ONLY);
ResultSet  rs  =  stmt.executeQuery(String sql);
```

4. 关闭数据库连接，释放资源

对数据库的访问结束后，及时关闭 ResultSet 对象、Statement 对象和 Connection 对象，从而释放所占的资源，实现代码与连接 SQL Server 一样。

【例 5-2】 在 JSP 中连接 Access 数据库。

本例创建一个 xs.mdb 数据库，然后在数据库中创建一个 XSB 数据表，这个数据表包

括 XH(学号)、XM(姓名)、XB(性别)、NL(年龄)和 ZY(专业)5 个字段,并设置 XH 为主键,XSB 数据表中各字段属性如图 5-7 所示,在数据表中输入学生的数据信息。

要通过 JDBC-ODBC 桥访问数据库,就必须先为数据库建立一个 ODBC 数据源,这样数据库才能实现和应用程序的交互。下面为 xs.mdb 数据库创建一个数据源,具体操作步骤如下。

XSB : 表

字段名称	数据类型
XH	文本
XM	文本
XB	文本
NL	文本
ZY	文本

图 5-7　XSB 表结构

(1) 选择“开始”|“控制面板”|“管理工具”|“数据源(ODBC)”选项,打开“ODBC 数据源管理器”对话框,如图 5-8 所示。



图 5-8　“ODBC 数据源管理器”对话框

(2) 单击“添加”按钮,打开“创建新数据源”对话框,如图 5-9 所示。



图 5-9　“创建新数据源”对话框

(3) 在对话框中选中 Microsoft Access Driver(*.mdb)选项,然后单击“完成”按钮,打开“ODBC Microsoft Access 安装”对话框,如图 5-10 所示。

(4) 在“数据源名”文本框中输入要创建的数据源名称,本例输入的数据源名 xs_access,然后单击“数据库”选项组中的“选择”按钮,打开“选择数据库”对话框,如图 5-11 所示。



图 5-10　“ODBC Microsoft Access 安装”对话框

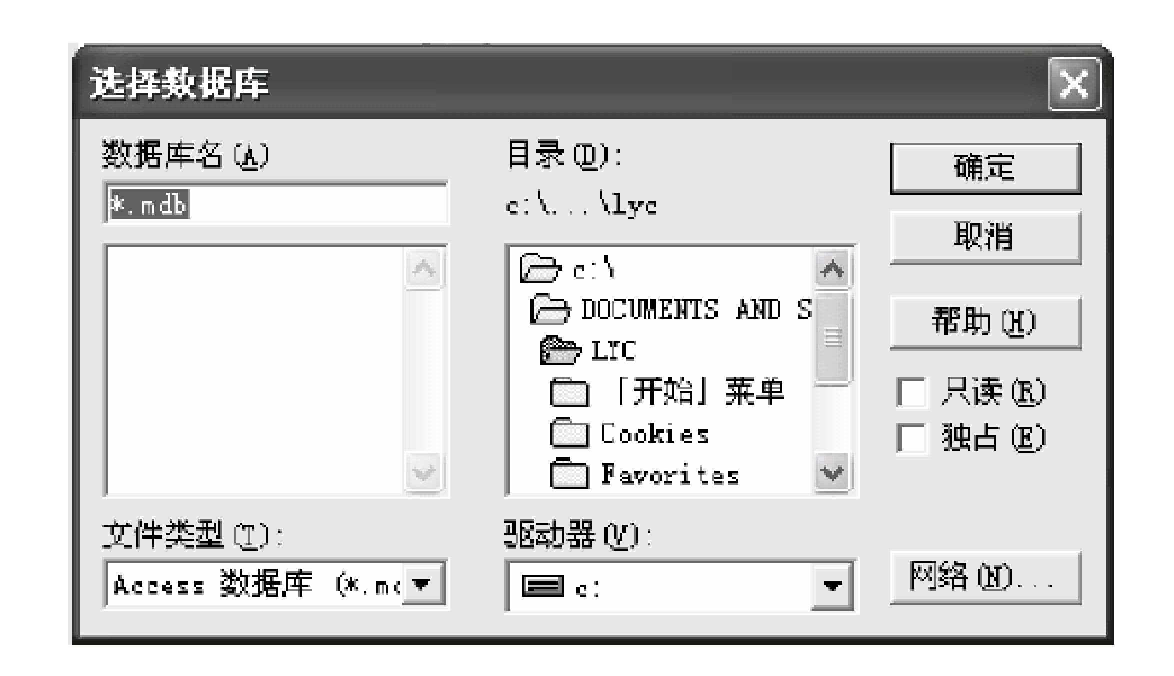


图 5-11　“选择数据库”对话框

(5) 选择前面创建的 xs.mdb 数据库，然后单击“确定”按钮，回到图 5-8 所示的对话框中，可以看到刚才创建的 xs_access 数据源已经出现在此对话框中，再单击“确定”按钮，数据源创建过程结束。

数据源创建结束之后，还需创建操作数据库的 Java 程序 AccessConn.java。该文件经过编译后形成.class 文件，保存在\Tomcat6.0\webapps\ROOT\WEB-INF\classes\XSCJ 目录下，需要手动建立 XSCJ 文件夹，该程序代码如下。本例源代码存放于本书配套素材中的 AccessConn.java 文件中(文件路径为 code\5)。

<!-- 例程 5-2 AccessConn.java -->

```
package XSCJ;
import java.sql.*;
public class AccessConn
{
    private Statement stmt = null;
    private Connection conn  = null;
    ResultSet rs = null;
    public AccessConn() { }
    public void OpenConn()throws Exception
    {
        try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            String url = "jdbc:odbc:xs_access";
            String user = "";
            String pwd = "";
            conn = DriverManager.getConnection(url,user,pwd);
        }
        catch(SQLException e){
            System.err.println("Data.executeQuery: " + e.getMessage());
        }
    }
    //执行查询类的 SQL 语句,有返回集
    public ResultSet executeQuery(String sql)
    {
```

```
            rs = null;
            try {
                stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                     ResultSet.CONCUR_READ_ONLY);
                 rs = stmt.executeQuery(sql);
            }
            catch(SQLException e) {
                System.err.println("Data.executeQuery: " + e.getMessage());
            }
            return rs;
        }
                //关闭对象
        public void closeStmt()
        {
            try {
                  stmt.close();
             }
            catch(SQLException e) {
                  System.err.println("Date.executeQuery: " + e.getMessage());
            }
        }
        public void closeConn()
        {
            try{
                  conn.close();
             }
            catch(SQLException e) {
                  System.err.println("Data.executeQuery: " + e.getMessage());
             }
        }
    }
```

将 AccessConnBean 创建好之后，编写 Access.jsp 文件，该文件是通过 AccessConn 访问 Access 数据库的 JSP 程序，程序源代码如下。本例源代码存放于本书配套素材中的 Access.jsp 文件中(文件路径为 code\5)。

<!-- 例程 5-3 Access.jsp -->

```
<%@ page contentType="text/html;charset=gb2312" %>
<%@ page language="java" import="java.sql.*" %>
<jsp:useBean id="AccessBean" scope="page" class="XSCJ_Bean.AccessConnBean" />
<HTML>
<HEAD>
<TITLE>使用 JDBC-ODBC 桥访问 Access 数据库</TITLE>
</HEAD>
<BODY>
<CENTER>
<Font size=5 color=blue>JDBC-ODBC 桥访问 Access 数据库</Font>
</CENTER><BR><HR><BR>
<Table border=2 bordercolor="#FFCCCC" align="center">
<tr bgcolor=CCCCCC align=center>
          <td><b>记录序列号</b></td>
```

```
                <td><b>学号</b></td>
                <td><b>姓名</b></td>
                <td><b>性别</b></td>
                <td><b>年龄</b></td>
                <td><b>专业</b></td>
</tr>
<%
        //查询 XSB 表中所有字段的记录
String sql = "select * from XSB";
    //调用 AccessBean 中加载驱动的成员函数 OpenConn()
AccessBean.OpenConn();
        ResultSet rs = AccessBean.executeQuery(sql);    //创建结果集
        while(rs.next())
{
%>
        <tr align = center>
            <td><% = rs.getRow() %></td>
            <td><% = rs.getString("XH") %></td>
            <td><% = rs.getString("XM") %></td>
            <td><% = rs.getString("XB") %></td>
            <td><% = rs.getString("NL") %></td>
            <td><% = rs.getString("ZY") %></td>
        </tr>
<%
}
%>
<%
out.print("数据库操作成功,恭喜你");
        rs.close();
        AccessBean.closeStmt();
        AccessBean.closeConn();
%>
</Table>
</CENTER>
</BODY>
</HTML>
```

在这段代码中，数据库查询的结果保存在 ResultSet 对象 rs 中。这里主要用到了 rs.next()方法、rs.getRow()方法和 rs.getString()方法。其中 rs.getRow()方法用来获取当前记录所在的行数，即当前记录是第几条记录。程序运行后的显示效果如图 5-12 所示。

JDBC-ODBC桥访问Access数据库

数据库操作成功，恭喜你

记录序列号	学号	姓名	性别	年龄	专业
1	06011201	贺芳芳	F	22	计算机科学与技术
2	06011202	张海	M	23	计算机科学与技术
3	06011203	李娜	F	21	计算机科学与技术
4	06011205	李宏	M	21	计算机科学与技术
5	06011206	何勇	F	25	计算机科学与技术
6	06011207	范世杰	F	24	计算机科学与技术
7	06011204	李霞	F	24	计算机科学与技术
8	06011208	黄晓晓	F	23	计算机科学与技术

图 5-12　access.jsp 运行效果

5.4　操作数据库

一般来说，数据库的操作分为两种：第一种是数据查询；第二种是数据更新。数据更新包括数据插入、修改和删除。

5.4.1 数据查询

在实际应用中，经常需要从数据库中查询某些特定的数据信息，如学号为 1002 的学生信息、地点在北京的仓库信息等。要实现指定数据信息的查询，需要给出一个参数，然后再利用 SQL 语句就可以将数据信息从数据库中查询出来。本节以查询指定学号的学生信息为例，介绍如何在 JSP 中实现数据的查询与定位。

首先利用 SQL Server 2000 创建一个数据库 student，并在该数据库在建立数据表 stu_info，其表结构如图 5-13 所示。在该表中输入如表 5-1 所示的学生信息。

列名	数据类型	长度	允许空
id	char	10	
name	char	10	✓
sex	char	2	✓
age	int	4	✓
dept	char	10	✓
grade	char	10	✓

图 5-13 stu_info 表结构

【例 5-3】 按学号查找学生信息。

本例包括两个 JSP 页面，即 find.jsp（用来输入要查询学生的学号）及 show.jsp（用来显示学生的详细信息），程序源代码如下。本例源代码存放于本书配套素材中的相应文件中（文件路径为 code\5）。

```
<!-- 例程 5-4 find.jsp -->

<%@ page contentType="text/html; charset=gb2312" language="java" import="java.sql.*"
errorPage="" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
    <title>查询数据</title>
    <style type="text/css">
    </style>
  </head>
  <body>
    <div align="center" class="style1">
    <p>请输入要查询学生的学号  </p>
    <form name="form1" method="post" action="show.jsp">
      <input type="text" name="id">
        
      <input type="submit" name="Submit" value="查询">
     </form>
    <p><span class="style2"></span></p>
    </div>
  </body>
</html>
```

```
<!-- 例程 5-5 show.jsp -->

<%@ page contentType="text/html; charset=gb2312" language="java" import="java.sql.*"
errorPage="" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/
```

```
loose.dtd">
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
  <title>查找某个学生的信息</title>
  <style type="text/css">
  </style>
</head>
  <body>
<%
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
String  url="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=student";
String  user="sa";
String  password="";
Connection  conn=  DriverManager.getConnection(url,user,password);
Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
String id=request.getParameter("id");
String sql="select * from stu_info where id="+id;
ResultSet rs = stmt.executeQuery(sql);//建立ResultSet(结果集)对象,并执行SQL语句
 %>
          <center class="style1">
            </div>
        <hr>
<%
//利用while循环配合next()方法将数据表中的记录列出
while(rs.next())
{
   session.setAttribute("id",rs.getString("id"));   //将ID保存到session中
 %>
             <p align="center">学号:<%=rs.getString("id")%></p>
             <p align="center">姓名:<%=rs.getString("name")%></p>
             <p align="center">性别:<%=rs.getString("sex")%></p>
             <p align="center">年龄:<%=rs.getString("age")%></p>
<%
}
rs.close();
stmt.close();
conn.close();
 %>
<p align="center"><input type="button"  id="back"name="back" value="返回"
 onClick="javascript:history.go(-1)"></p>
 </body>
</html>
```

本程序在 find.jsp 页面中输入要查询的学生学号后，通过表单将其提交到 show.jsp 页面中，show.jsp 页面根据这个参数查询出该学生的详细信息并输出到页面。程序的运行效果如图 5-14 和图 5-15 所示。

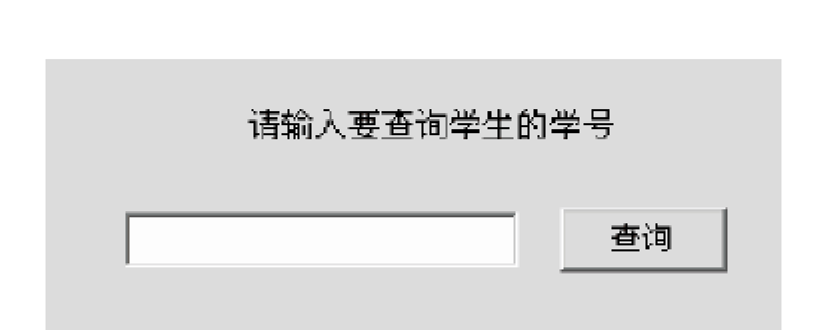


图 5-14　find.jsp 页面运行效果



图 5-15　show.jsp 运行效果

5.4.2　数据更新

数据更新操作包括修改数据、添加数据、删除数据。在数据库的维护过程中经常需要修改数据表中的记录信息，如在学生管理系统中修改学生的姓名、年龄等。通常是通过以下两个步骤来实现数据的更新的。

1. 创建语句对象

```
Statement stmt = conn.createStatement(int type, int  concurrency );
```

2. 执行更新

```
String sql = "sqlStatement " ; //插入或修改或删除 SQL 字符串
int  number = stmt.executeUpdate(sql);     //执行更新操作
```

本节依然以 student 数据库中的 stu_info 表为例，介绍在 JSP 程序中如何修改数据库中的记录信息。

【例 5-4】 修改学生表信息。

在此例中，学生的 ID(即学号)没有变化，要修改的是学生的姓名、性别和年龄。该程序包括 3 个 JSP 页面，分别为 modi_1.jsp、modi_2.jsp 和 modi_3.jsp。程序运行时，用户首先要在 modi_1.jsp 页面中选择要修改的学生学号，该程序源代码如下。本例源代码存放于本书配套素材中的相应文件中(文件路径为 code\5)。

<!-- 例程 5 - 6 modi_1.jsp -->

```
<%@ page contentType = "text/html; charset = gb2312" language = "java" import = "java.sql.*"
errorPage = "" %>
<!DOCTYPE HTML PUBLIC " - //W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
 <head>
  <meta http - equiv = "Content - Type" content = "text/html; charset = gb2312">
  <title>修改学生信息</title>
  <style type = "text/css">
  </style>
```

```
 </head>
 <body>
<%
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
String  url = "jdbc:microsoft:sqlserver://localhost:1433;DatabaseName = student";
String  user = "sa";
String  password = "";
Connection  conn =  DriverManager.getConnection(url,user,password);
//建立 Statement 对象
Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
%>
 <center><h2 class = "style3">修改学生信息</h2>
 <form name = "form1" method = "post" action = "modi_2.jsp">
 <p><span class = "style4">请选择要修改的学生的学号:</span>
 <select name = "id">
 <% ResultSet rs = stmt.executeQuery("SELECT * FROM stu_info");
  while(rs.next())
  {
    String id = rs.getString("id");
    %>
    <option value = "<% = id %>"><% = id %></option>
    <% } %>
  }
  </select>
  </p>
  <p>
   <input type = "submit"  name = "queding" value = "确定">
    
   <input name = "back" type = "button" id = "back" value = "返回"
     onClick = "javascript:history.go( - 1)">
  </p>
  </form>
 </center>
</body>
</html>
```

该程序的运行效果如图 5-16 所示。



图 5-16　modi_1.jsp 的运行效果

在该页面中选择学号后，单击“确定”按钮，通过表单将参数 ID 提交给 modi_2.jsp 页面。在 modi_2.jsp 页面中根据参数 ID 取出整条记录信息并显示出来，用户可以输入新的姓名、性别及年龄信息。modi_2.jsp 程序的源代码如下。

<!-- 例程 5 - 7 modi_2.jsp -->

```
<% @ page contentType = "text/html; charset = gb2312" language = "java" import = "java.sql. * "
errorPage = ""  %>
<! DOCTYPE HTML PUBLIC " - //W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
  <title>修改学生信息</title>
  <style type="text/css">
 </style>
</head>
<body>
 <div align="center" class="style6">
  <span class="style6">以下是该学生的信息</span><BR>
 </div>
<BR>
<table border=2  align="center">
   <tr>
      <td> id</td>
      <td> name</td>
      <td> sex</td>
      <td> age</td>
   </tr>
<%
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
String  url="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=student";
String  user="sa";
String  password="";
Connection  conn=  DriverManager.getConnection(url,user,password);
Statement  stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_
UPDATABLE);
String id=request.getParameter("id");
String sql="select * from stu_info where id="+id;
ResultSet  rs=stmt.executeQuery(sql);
while(rs.next())  {
   session.setAttribute("id",rs.getString("id"));   //将 ID 保存到 session 中
   %>
   <tr>
     <td><%=rs.getString("id")%></td>
     <td><%=rs.getString("name")%></td>
     <td><%=rs.getString("sex")%></td>
     <td><%=rs.getString("age")%></td>
   </tr>
<%}%>
<%
   rs.close();
   stmt.close();
   conn.close();
 %>
</table>
<hr>
<p align="center" class="style6">请输入新的学生信息</p>
<form name="form1" method="post" action="modi_3.jsp">
   <div align="center">
      <p> </p>
```

```
        <p> name:
          < input type = "text" name = "name">
        </p>
        <p> sex:
          < input type = "text" name = "sex">
        </p>
        <p> age:
          < input type = "text" name = "age">
        </p>
        <p>
          < input type = "submit" name = "Submit" value = "修改">
           
          < input name = "Submit" type = "reset" value = "取消">
           
       </p>
    </div>
</form>
<p align = "center" class = "style6">  </p>
</body>
</html>
```

该程序的运行效果如图 5-17 所示。

以下是该学生的信息

id	name	sex	age
95004	王小萌	女	19

请输入新的学生信息

name

sex

age

修改 取消

图 5-17 modi_2.jsp 的运行效果

在该页面中输入新的学生信息后，需要提交到 modi_3.jsp 页面中。在 modi_3.jsp 页面中先调用 ISOtoGB 类文件将输入的文本格式转换为符合要求的 GB2312 格式，然后再将 stu_info 表中的数据更新。

ISOtoGB.java 类文件的源代码如下。

<!-- 例程 5-8 ISOtoGB.java -->

```
package modify;
public class ISOtoGB{
public static String convert(String str) {
          try {
```

```
                byte[] bytesStr = str.getBytes("ISO-8859-1");
                return new String(bytesStr,"gb2312");
            } catch(Exception ex) {
                return str;
            }
        }
}
```

modi_3.jsp 程序源代码如下。

```
<!-- 例程 5-9 modi_3.jsp -->

<%@ page contentType="text/html; charset=gb2312" language="java" import="java.sql.*"
errorPage="" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/
loose.dtd">
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
        <title>修改学生信息</title>
        <style type="text/css">
        </style>
    </head>
    <body>
<jsp:useBean id="ISO_GB" scope="page" class="modify.ISOtoGB" />
<%
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
String  url="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=student";
String  user="sa";
String  password="";
Connection  conn=  DriverManager.getConnection(url,user,password);
Statement stmt = conn.createStatemen(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_
UPDATA);
Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_
READ_ONLY);
String id=(String)session.getAttribute("id");   //取出 ID 参数
String name=ISO_GB.convert(request.getParameter("name"));
String sex=request.getParameter("sex");   //获取 sex 参数
String age=request.getParameter("age");   //获取 age 参数
String sql ="update stu_info set name='"+name+"',sex='"+sex+"',age='"+age+"'where id
="+id;
stmt.executeUpdate(sql);     //执行 SQL 语句
stmt.close();                //关闭 Statement 对象
conn.close();                //关闭 Connection 对象
%>
 <center><h2 class="style1">修改学生信息</h2>
 <p><br>
 <span class="style3"><span class="style4">记录修改成功</span>!</span></p>
 <form name="form1" method="post" action="modi_1.jsp">
```

```
    <input type="submit"  id="back" name="back" value="返回">
  </form>
  <p> </p>
  </center>
</body>
</html>
```

modi_3.jsp 程序的运行效果如图 5-18 所示。

图 5-18 modi_3.jsp 的运行效果

5.4.3 数据删除

在数据库的维护过程中,经常要删除一些已经没用的记录。本节介绍在 JSP 程序中如何删除数据库中的记录。

【例 5-5】 删除学生表 stu_info 中的记录。

本例包括 3 个 JSP 页面,程序运行时首先在 del_1.jsp 页面中选择要删除的学号,程序将选择的 ID 参数提交到 del_2.jsp 页面,此页面根据传过来的 ID 参数取出该学生的详细信息记录,确定确实要删除该学生信息后,将 ID 参数提交到 del_3.jsp 页面,最后由 del_3.jsp 程序使用 delete 语句删除记录。该程序 3 个页面的源代码如下。本例源代码存放于本书配套素材中的相应文件中(文件路径为 code\5)。

```
<!-- 例程 5-10 del_1.jsp -->

<%@ page contentType="text/html; charset=gb2312" language="java" import="java.sql.*"
errorPage="" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/
loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
        <title>删除学生信息</title>
        <style type="text/css">
        </style>
    </head>
    <body>
<%
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
String  url="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=student";
String  user="sa";
String  password="";
Connection  conn=  DriverManager.getConnection(url,user,password);
Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
 %>
 <center><h2 class="style3">删除学生信息</h2>
  <br>
   <form name="form1" method="post" action="del_2.jsp">
     <p><span class="style4">请选择要删除的学生的学号:</span>
      <select name="id">
```

```
        <% ResultSet rs = stmt.executeQuery("SELECT * FROM stu_info");
   while(rs.next())
   {
   String id = rs.getString("id");
   %>
  <option value = "<% = id %>"><% = id %></option>
   <% } %>
   }
   </select>
   </p>
  <p>
  <input type = "submit"   name = "queding" value = "确定">
    
  <input name = "back" type = "button" id = "back" value = "返回"
   onClick = "javascript:history.go( - 1)">
  </p>
  </form>
</center>
</body>
</html>
```

该程序运行后的效果如图 5-19 所示。

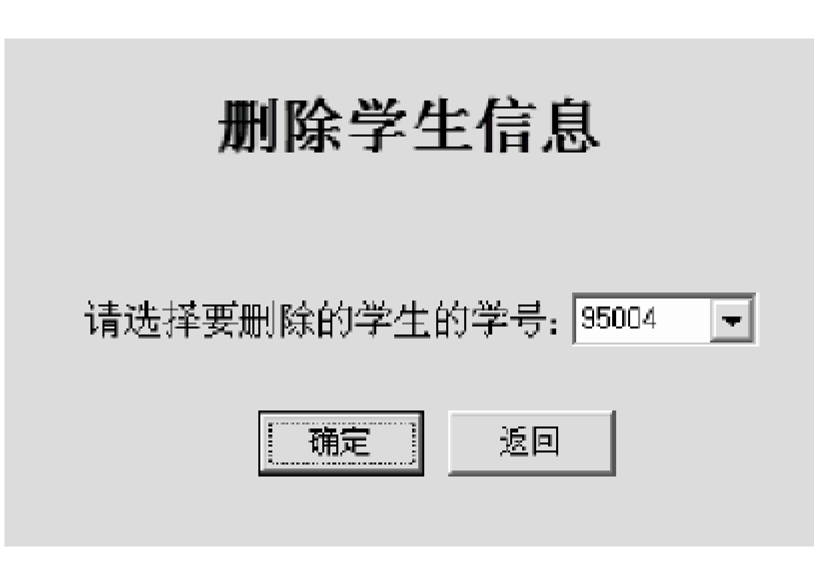


图 5-19 del_1.jsp 的运行效果

<!-- 例程 5 - 11 del_2.jsp -->

```
<% @ page contentType = "text/html; charset = gb2312" language = "java" import = "java.sql. * "
errorPage = ""  %>
<! DOCTYPE HTML PUBLIC " - //W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/
loose.dtd">
<html>
    <head>
    <meta http - equiv = "Content - Type" content = "text/html; charset = gb2312">
    <title>删除学生信息</title>
    <style type = "text/css">
    </style></head>
    <body>
        <CENTER>
        <div align = "center">
<%
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
String  url = "jdbc:microsoft:sqlserver://localhost:1433;DatabaseName = student";
String  user = "sa";
String  password = "";
Connection  conn =  DriverManager.getConnection(url,user,password);
Statement stmt  = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
String id = request.getParameter("id");
String sql = "select * from stu_info where id = " + id;
ResultSet rs  = stmt.executeQuery(sql);
//建立 ResultSet(结果集)对象,并执行 SQL 语句
 %>
 <center class = "style1">
 <span class = "style2">确实要删除这个学生的信息吗?</span>
```

```
</div>
<hr>
<form name="form1" method="post" action="del_3.jsp">
 <div align="center">
<%
while(rs.next())
{
    session.setAttribute("id",rs.getString("id"));//将 ID 保存到 session 中
%>
 </div>
            <p align="center">学号:<% =rs.getString("id")%></p>
            <p align="center">姓名:<% =rs.getString("name")%></p>
            <p align="center">性别:<% =rs.getString("sex")%></p>
            <p align="center">年龄:<% =rs.getString("age")%></p>
            <div align="center">
<%
}
rs.close(); //关闭 ResultSet 对象
stmt.close(); //关闭 Statement 对象
conn.close(); //关闭 Connection 对象
%>
<input name="queding" type="submit" id="confirm" value="确定">
  
 <input type="reset" name="back" id="back" value="返回">
</div>
</form>
</body>
</html>
```

在该程序代码中,首先使用 JDBC 驱动程序连接数据库,然后使用 Request 对象的 getParameter()方法来获取 del_1.jsp 页面传过来的参数 ID,再根据这个参数使用 select 语句将该学生的详细信息记录显示,同时将这个 ID 保存在 Session 对象中。该程序运行后的效果如图 5-20 所示。

确实要删除这个学生的信息吗?

学号: 95004

姓名: 王小萌

性别: 女

年龄: 19

确定　返回

图 5-20　del_2.jsp 程序的运行效果

```
<!-- 例程 5-12 del_3.jsp -->

<%@ page contentType="text/html; charset=gb2312" language="java" import="java.sql.*"
errorPage="" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/
loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
        <title>删除学生信息</title>
        <style type="text/css">
        </style>
    </head>
    <body>
<%
```

```
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
String  url = "jdbc:microsoft:sqlserver://localhost:1433;DatabaseName = student";
String  user = "sa";
String  password = "";
Connection  conn =  DriverManager.getConnection(url,user,password);
Statement stmt  = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
String id = (String)session.getAttribute("id");                //取出 ID 参数
String sql = "delete  from stu_info where id = " + id;         //删除符合条件的记录
stmt.executeUpdate(sql);        //执行 SQL 语句
stmt.close();     //关闭 Statement 对象
conn.close();     //关闭 Connection 对象
%>
 <center><h2 class = "style1">删除学生信息</h2>
 <p><br>
 <span class = "style3">记录删除成功!</span></p>
 <form name = "form1" method = "post" action = "index.htm">
    <input type = "submit"  id = "back" name = "back" value = "返回">
 </form>
 <p> </p>
</center>
</body>
</html>
```

在该程序的代码中，使用 Session 对象的 getAttribute()方法获取 del_2.jsp 页面中传来的 ID 参数，并根据这个参数使用 delete 语句实现记录的删除。该程序的运行效果如图 5-21 所示。

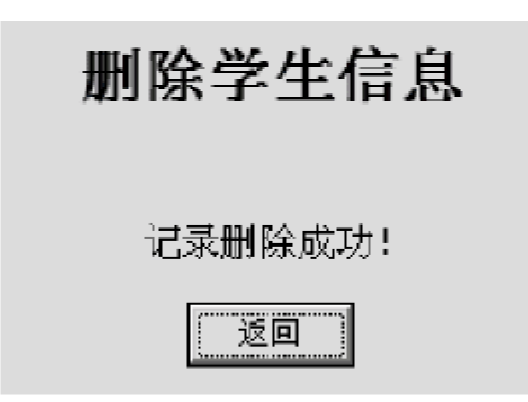


图 5-21　del_3.jsp 页面运行效果

5.5　上机指导与练习

5.5.1　查询英语成绩及格的学生信息

1. 练习目标

（1）掌握数据表的创建。

（2）熟悉数据查询的方法。

2. 练习指导

（1）首先利用 Access 建立数据库表 students，其中表结构如图 5-22 所示。

字段名称	数据类型	说明
number	文本	学号
name	文本	姓名
math	数字	数学成绩
english	数字	英语成绩
phics	数字	物理成绩

图 5-22　students 表结构

（2）按例 5-2 所示介绍的步骤加载驱动程序，获取连接对象。

（3）编写程序代码 grade.jsp。在本例中，语句对象 sql 是通过无参的 createStatement()方法创建的，该程序源代码如下。

```
<!-- 例程 5 - 13 grade.jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<%@ page import = "java.sql.*" %>
<HTML>
<BODY>
<center>
  <%
     Connection con;
     Statement sql;
     ResultSet rs;
     try{
           Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        }
     catch(ClassNotFoundException e) {}
     try{
           con = DriverManager.getConnection("jdbc:odbc:grade" );
           sql = con.createStatement();
           rs = sql.executeQuery("select * from students where english>= 60");
           out.print("<Table Border = 1 bgcolor = cyan >");
           out.print("<TR>");
           out.print("<TH width = 50 >" + "学号");
           out.print("<TH width = 50 >" + "姓名");
           out.print("<TH width = 100 >" + "数学成绩");
           out.print("<TH width = 100 >" + "英语成绩");
           out.print("<TH width = 100 >" + "物理成绩");
           out.print("</TR>");
           while(rs.next())
              {
                out.print("<TR>");
                out.print("<TD >" + rs.getString(1) + "</TD>");
                out.print("<TD >" + rs.getString(2) + "</TD>");
                out.print("<TD >" + rs.getInt(3) + "</TD>");
                out.print("<TD >" + rs.getInt(4) + "</TD>");
                out.print("<TD >" + rs.getInt(5) + "</TD>");
                out.print("</TR>") ;
              }
          out.print("</Table>");
          con.close();
       }
     catch(SQLException e1)  {}
  %>
</center>
</body>
</html>
```

本例源代码存放于本书配套素材中的相应文件中(文件路径为 code\上机指导\5)。

5.5.2 向表中添加记录

1. 练习目标

(1) 熟悉数据添加的方法。
(2) 熟悉驱动程序的加载。

2. 练习指导

本程序是向 students 表中添加记录,该程序由两个页面组成,表单界面 insert.jsp 将录入的数据提交给 newdata.jsp 页面,newdata.jsp 页面负责将数据添加到表 students 中并显示添加后的记录。该程序的源代码如下。

```
<!-- 例程 5-14 insert.jsp -->

<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.sql.*" %>
<HTML>
<BODY  bgcolor=pink  >
<FONT  size=3>
    <p><b>数据录入界面</b>
    <FORM  action="newDatabase.jsp" method=post>
        学号:<Input type="text" name="number"><BR>
        姓名:<Input type="text" name="name">  <BR>
        数学成绩:<Input type="text" name="math">  <BR>
        英语成绩:<Input type="text" name="english"><BR>
        物理成绩:<Input type="text" name="physics"><br>
        <Input type="submit" name="b" value="添加记录">
    </FORM>
    <p>添加记录前的表
  <%
     String name,number;
     int math,physics,english;
     Connection con;
     Statement sql;
     ResultSet rs;
     try{
           Class.forName("sun.jdbc.odbc.jdbcodbcDriver");
        }
     catch(ClassNotFoundException e)
        {}
     try {
            con=DriverManager.getConnection("jdbc:odbc:grade","","");
            sql=con.createStatement();
            rs=sql.executeQuery("SELECT * FROM students");
            out.print("<Table Border>");
            out.print("<TR>");
            out.print("<TH width=100>"+"学号"+"</TH>");
```

```
            out.print("<TH width=100>"+"姓名"+"</TH>");
            out.print("<TH width=50>"+"数学成绩"+"</TH>");
            out.print("<TH width=50>"+"英语成绩"+"</TH>");
            out.print("<TH width=50>"+"物理成绩"+"</TH>");
            out.print("</TR>");
            while(rs.next())
              {
                out.print("<TR>");
                number=rs.getString(1);
                out.print("<TD>"+number+"</TD>");
                name=rs.getString(2);
                out.print("<TD>"+name+"</TD>");
                math=rs.getInt(3);
                out.print("<TD>"+math+"</TD>");
                english=rs.getInt(4);
                out.print("<TD>"+english+"</TD>");
                physics=rs.getInt(5);
                out.print("<TD>"+physics+"</TD>");
                out.print("</TR>");
              }
            out.print("</Table>");
            con.close();
        }
      catch(SQLException e1) {}
   %>
</FONT>
</BODY>
</HTML>
```

```
<!-- 例程 5-15 newdata.jsp -->
```

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.sql.*" %>
<HTML>
<BODY bgcolor=pink>
<FONT size=3>
  <% //获取提交的学号
     String number=request.getParameter("number");
     if(number==null)
       {
         number="";
       }
     byte b[]=number.getBytes("ISO-8859-1");
     number=new String(b);
     //获取提交的姓名
     String name=request.getParameter("name");
     if(name==null)
       {
         name="";
       }
     byte c[]=name.getBytes("ISO-8859-1");
```

```
        name = new String(c);
        //获取提交的新的数学成绩
        String m = request.getParameter("math");
        if(m == null)
          {
            m = " - 100";
          }
         //获取提交的新的英语成绩
         String e = request.getParameter("english");
         if(e == null)
            {
                e = " - 100";
            }
         //获取提交的新的物理成绩
         String p = request.getParameter("phics");
         if(p == null)
            {
              p = " - 100";
            }
         Connection con = null;
         Statement sql = null;
         ResultSet rs = null;
         try{
              Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            }
         catch(ClassNotFoundException event){}
         try{
              con = DriverManager.getConnection("jdbc:odbc:grade","","");
              sql = con.createStatement();
              String condition =
"INSERT INTO students VALUES" + "(" + "'" + number + "','" + name + "'," + m + "," + e + "," + p + ")";
              sql.executeUpdate(condition);     //执行添加操作
   %>
   <p>添加记录后的表
   <%
              rs = sql.executeQuery("SELECT * FROM students ORDER BY number ");
              out.print("<Table Border>");
              out.print("<TR>");
              out.print("<TH width = 100>" + "学号" + "</TH>");
              out.print("<TH width = 100>" + "姓名" + "</TH>");
              out.print("<TH width = 50>" + "数学成绩" + "</TH>");
              out.print("<TH width = 50>" + "英语成绩" + "</TH>");
              out.print("<TH width = 50>" + "物理成绩" + "</TH>");
              out.print("</TR>");
              while(rs.next())
                {
                  out.print("<TR>");
                  String n = rs.getString(1);
                  out.print("<TD>" + n + "</TD>");
                  String xingming = rs.getString(2);
                  out.print("<TD>" + xingming + "</TD>");
```

```
                int math = rs.getInt(3);
                out.print("<TD>" + math + "</TD>");
                int english = rs.getInt(4);
                out.print("<TD>" + english + "</TD>");
                int phics = rs.getInt(5);
                out.print("<TD>" + physics + "</TD>");
                out.print("</TR>");
              }
            out.print("</Table>");
            con.close();
          }
      catch(SQLException event)
          {}
    %>
  </FONT>
  </BODY>
  </HTML>
```

本例源代码存放于本书配套素材中的相应文件中(文件路径为 code\上机指导\5)。

5.5.3　网上投票系统

1. 练习目标

(1) 熟悉连接数据库的操作。
(2) 熟悉数据库的查询与更新操作。

2. 练习指导

(1) 创建两个表,一个是 IP 表,表结构为(IP),该表用来存放投票人的 IP 地址。另一个是 candidate 表,表结构为(name,count),该表用来存放候选人的名单及候选人的得票数。

(2) 建立一个 toupiao.jsp 程序,用于展示投票界面,该程序文件的源代码如下。

```
<!-- 例程 5-16 toupiao.jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<%@ page import = "java.sql.*" %>
<HTML>
<BODY>
  <%
     StringBuffer nameList = new StringBuffer();
     Connection con;
     Statement sql;
     ResultSet rs;
     try{
           Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
         }
     catch(ClassNotFoundException e){}
     try{
           con = DriverManager.getConnection("jdbc:odbc:grade","","");
```

```
            sql = con.createStatement();
            rs = sql.executeQuery("SELECT * FROM candidate");
            nameList.append("<FORM action = vote.jsp Method = post>");
            nameList.append("<Table Border>");
            nameList.append("<TR>");
            nameList.append("<TH width = 100>" + "姓名" + "</TH>");
            nameList.append("<TH width = 50>" + "投票选择" + "</TH>");
            nameList.append("</TR>");
            while(rs.next())  //取出表 candidate 中的数据,以 table 格式显示在表单中
              {
                  nameList.append("<TR>");
                  String name = rs.getString(1);
                  nameList.append("<TD>" + name + "</TD>");
                  String s = "<Input Type = radio name = name value = " + name + " >";
                  nameList.append("<TD>" + s + "</TD>");
                  nameList.append("</TR>");
              }
            nameList.append("</Table>");
            nameList.append("<Input Type = submit value = 投票>");
            nameList.append("</FORM ");
            con.close();
            out.print(nameList);
          }
        catch(SQLException el) {}
      %>
       <br></br>
       <FORM action = "showvote.jsp" method = post name = form2>
          <Input type = submit name = "g" value = "查看投票情况">
       </Form>
    </BODY>
    </HTML>
```

(3) 建立第二个界面 vote.jsp,用于将客户投票的选择保存到 candidate 表中,实现投票统计,并将客户的 IP 地址保存到 IP 表中。该程序源代码如下。

<!-- 例程 5-17 vote.jsp -->

```
<%@ page contentType = "text/html;charset = GB2312" %>
<%@ page import = "java.sql.*" %>
<%@ page import = "java.io.*" %>
<HTML>
<BODY>
  <%!
      int total = 0;                          //记录总票数的变量
      synchronized void countTotal()          //操作总票数的同步方法
      {
          total ++ ;
      }
   %>
  <%
      Connection con = null;
```

```
    Statement sql = null;
    ResultSet rs = null;
    boolean vote = true;                //决定用户是否有权投票的变量
    String name = "";                   //得到被选择的候选人名字
    name = request.getParameter("name");
    if(name == null)
      {
        name = "?";
      }
  byte a[ ] = name.getBytes("ISO - 8859 - 1");
  name = new String(a);
  String IP = (String)request.getRemoteAddr();            //得到投票人的 IP 地址
  try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");    //加载驱动程序
    }
  catch(ClassNotFoundException e)
    {}
  //首先查询 IP 表,判断投票客户的 IP 地址是否已经投过票
  try{
        con = DriverManager.getConnection("jdbc:odbc:grade","","");
        sql = con.createStatement();
        rs = sql.executeQuery("select * from ip where ip = " + "'" + IP + "'");
        int row = 0;
        while(rs.next())
          {
              row++;
          }
        if(row >= 1)
          {
            vote = false;   //不允许投票
          }
    }
  catch(SQLException e)
    {}
  if(name.equals("?"))
    {
        out.print("您没有投票,没有权利看选举结果");
    }
  else
    {
        if(vote)
          {
            out.print("您投了一票");
            countTotal(); //将总票数加 1
            //给该候选人增加一票,同时将自己的 IP 地址写入数据库
            try
              {
          rs = sql.executeQuery("SELECT * FROM candidate WHERE name = " + "'" + name + "'");
                rs.next();
                int count = rs.getInt("count");
                count++;
                String condition =
               "update candidate set count = " + count + " where name = " + "'" + name + "'";
```

```
                    //执行更新操作(投票计数)
                    sql.executeUpdate(condition);
                    //将 IP 地址写入 IP 表
                    String to =
                    "insert into ip values" + "(" + "'" + IP + "'" + ")";
                    sql.executeUpdate(to);
                }
            catch(SQLException e)
                {
                    out.print("" + e);
                }
               //显示投票后表中的记录
             try{
                    rs = sql.executeQuery("SELECT * FROM candidate ");
                    out.print("<Table Border>");
                    out.print("<TR>");
                    out.print("<TH width = 100>" + "姓名" + "</TH>");
                    out.print("<TH width = 100>" + "得票数" + "</TH>");
                    out.print("<TH width = 100>" + "总票数:" + "</TH>");
                    out.print("</TR>");
                    while(rs.next())
                        {
                          out.print("<TR>");
                          out.print("<TD>" + rs.getString(1) + "</TD>");
                          int count = rs.getInt("count");
                          out.print("<TD>" + count + "</TD>");
                          double b = (count * 100)/total;     //得票的百分比
                          out.print("<TD>" + b + " % " + "</TD>");
                          out.print("</TR>");
                        }
                    out.print("</Table>");
                    con.close();
                 }
             catch(SQLException e)
                 {}
            }
        else
            {
              out.print("您已经投过票了");
            }
        }
   %>
</BODY>
</HTML>
```

(4) 建立第三个界面 showvote.jsp,用于显示投票结果。该程序源代码如下。

```
<!-- 例程 5-18 showvote.jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<%@ page import = "java.sql.*" %>
<%@ page import = "java.io.*" %>
<HTML>
<BODY>
```

```
    <p><font size=5>投票情况表</font>
    <%
       //加载桥接器
       try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
          }
       catch(ClassNotFoundException e)
          {}
       Connection con=null;
       Statement sql=null;
       ResultSet rs=null;
       ResultSet rs2=null;
       //显示投票后表中的记录
       try{
             con=DriverManager.getConnection("jdbc:odbc:grade","","");
             sql=con.createStatement();
             rs2=sql.executeQuery("select  sum(count) from candidate");
             rs2.next();
             int total=rs2.getInt(1);
             rs=sql.executeQuery("SELECT *  FROM candidate ");
             out.print("<Table Border>");
             out.print("<TR>");
             out.print("<TH width=100>"+"姓名"+"</TH>");
             out.print("<TH width=100>"+"得票数"+"</TH>");
             out.print("<TH width=100>"+"总票数:"+total+"</TH>");
             out.print("</TR>");
             while(rs.next())
                 {
                     out.print("<TR>");
                     out.print("<TD>"+rs.getString(1)+"</TD>");
                     int count=rs.getInt("count");
                     out.print("<TD>"+count+"</TD>");
                     double b=(count*100)/total;        //得票的百分比
                     out.print("<TD>"+b+"%"+"</TD>");
                     out.print("</TR>");
                 }
             out.print("</Table>");
             con.close();
          }
       catch(SQLException e)
          {}
    %>
  </BODY>
  </HTML>
```

本例源代码存放于本书配套素材中的相应文件中(文件路径为 code\上机指导\5)。

本章小结

本章介绍了页面与数据库之间的通信。通过实例讲解了 JSP 页面对数据库的主要操作,包括数据库的各种查询、数据录入、修改、删除等操作。通过本章的学习,可以掌握 JSP

及数据库的连接方法和基本操作。

习题 5

一、简答题

1. 简述 JDBC 的体系结构。
2. JDBC 的驱动程序有哪几类?

二、操作编程题

1. 完成 JDBC 与 SQL Server 的数据库连接。
2. 编写一个 JSP 页面,用于查询学生成绩。

第6章 文件操作

文件可以永久地存储信息，从本质上讲，文件就是存放在存储设备上的一系列数据的集合。应用程序如果想长期保存数据，就必须将数据存储到文件中，这就涉及文件的操作。本章主要介绍页面与文件之间的数据传输操作。

本章主要内容：

- 数据流与 File 类；
- 随机访问类的应用；
- 文件操作的应用。

6.1 File 类与数据流

众所周知，多数程序在不获取外部数据的情况下不能顺利完成目标，数据需要从一个输入源获得，而程序的结果则被送到输出目的地。

6.1.1 数据流

使用数据流可以读文件或写文件。所谓流，是一个生产或消费信息的逻辑实体，通过输入/输出系统与物理设备相连。虽然与之相连的物理设备各不相同，但所有的流都以同样的方式运转。

按照数据流动方向，可将数据流分为输入流和输出流，输入流只能读文件不能写文件，输出流只能写文件不能读文件。

数据流的模型如图 6-1 所示。

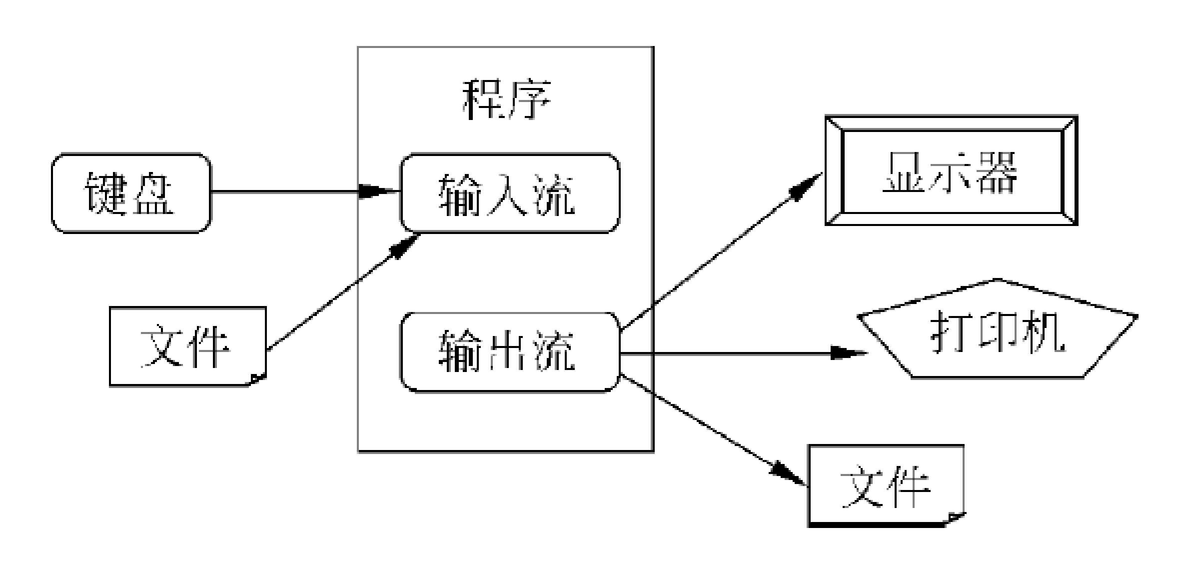


图 6-1 数据流模型

在程序中，使用输入流可从键盘或从文件中读取数据；使用输出流可向显示器、打印机或文件中传输数据。

6.1.2 File 类

除了一些流式操作的类外，还有一些用于文件系统操作的类，即 File 类。数据流类实现文件的顺序读写，File 类则直接处理文件和文件系统。使用 File 类可以访问文件属性信息，但不提供读/写文件的方法。另外，File 还浏览子目录层次结构，可以用来生成 File 对象的构造函数有如下 3 个。

(1) File(String directoryPath)：通过全路径——路径文件名来创建对象，路径可以是绝对路径也可以是相对路径。

(2) File(String directoryPath, String filename)：通过父目录和文件名来创建对象，filename 是不含路径的文件名。

(3) File(File f, String filename)：也是通过父目录和文件名来创建对象的，但父目录由一个 File 对象提供。

其中，filename 是文件名，directoryPath 是文件的路径名，f 是一个指定目录的文件对象。

专家点拨： 文件的路径有两种形式，即绝对路径和相对路径。绝对路径包含它所指定的文件的完整路径信息，根据绝对路径就可以唯一定位一个文件。相对路径是针对"其他某个路径"而言的，这个路径和相对路径共同定位一个文件的位置。

File 定义了很多获取 File 对象标准属性的方法。其实用方法如下。

1. 属性操作

(1) public String getName()：获取文件名。

(2) public String getPath()：获取文件路径。

(3) public String getAbsolutePath()：获取文件绝对路径。

(4) public long length()：获取文件的长度(单位是字节)。

(5) public String getParent()：获取文件的父目录。

(6) public File getParentFile()：获取文件父目录中的文件。

(7) public long lastModified()：获取文件最后修改时间(时间是从 1970 年午夜至文件最后修改时刻的毫秒数)。

(8) public boolean canRead()：判断文件是否可读。

(9) public boolean canWrite()：判断文件是否可被写入。

(10) public boolean exits()：判断文件是否存在。

(11) public boolean isFile()：判断文件是不是一个正常文件。

(12) public boolean isDirectroy()：判断文件是不是一个目录。

(13) public boolean isHidden()：判断文件是不是隐藏文件。

2. 文件操作

(1) public boolean renameTo(File dest)：给文件换名。

(2) public boolean delete()：删除文件。

3. 目录操作

(1) public boolean mkdir()：创建目录。
(2) public String[] list()：以字符串形式列出目录。
(3) public File[] listFiles()：以 File 对象形式列出目录。

【例 6-1】 获得文件信息。

本例在\code\6\test 目录下，创建了一个文件 newfile.txt，然后测试该文件的属性。该程序源代码如下。本程序源代码存放于本书配套素材中的相应文件中(文件路径为 code\6\)。

```
<!-- 例程 6-1 fileinfo.jsp -->
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.io.*" %>
<html>
  <body bgcolor=cyan>
    <font size=3>
      <%
          File f1 = new File("e:/code/6/test", "newfile.txt");
          f1.createNewFile(); //创建文件 newfile.txt
       %>
        <p>文件 newfile.txt 存在吗?:
          <% =f1.exists() %><BR>
        <P>newfile.txt 的父目录是:
        <% =f1.getParent() %><BR>
        <P>文件 newfile.txt 是可读的吗?
        <% =f1.canRead() %><BR>
        <P>文件 newfile.txt 的长度:
        <% =f1.length() %>字节 <BR>
      </font>
    </body>
</html>
```

本例调用 File 对象的 createNewFile()方法来创建一个文件，再调用 File 对象的相关方法获取文件属性。该程序的运行效果如图 6-2 所示。

文件newfile.txt存在吗?: true
newfile.txt的父目录是: e:\code\6\test
文件newfile.txt是可读的吗? true
文件newfile.txt的长度: 974字节

图 6-2　fileinfo.jsp 的运行效果

6.2 数据流成分

按照数据流成分划分，可将数据流分为字节流、字符流、缓冲流、数据流、对象流等。

6.2.1 字节流

字节流类为处理字节式输入/输出提供了丰富的环境，其处理单元为 1 个字节，操作字节和字节数组。一个字节流可以和其他任何类型的对象并用，包括二进制数据，这样的多功能性使得字节流对很多类型的程序都很重要。

字节流有两个超类，也是两个抽象类，分别是字节输入流(InputStream)和字节输出流(OutputStream)。

1. InputStream 类

该类是所有字节输入流的超类，是一个定义了流式字节输入模式的抽象类，该类的所有方法在出错条件下引发一个 IOException 异常。

InputStream 类层次图如图 6-3 所示。

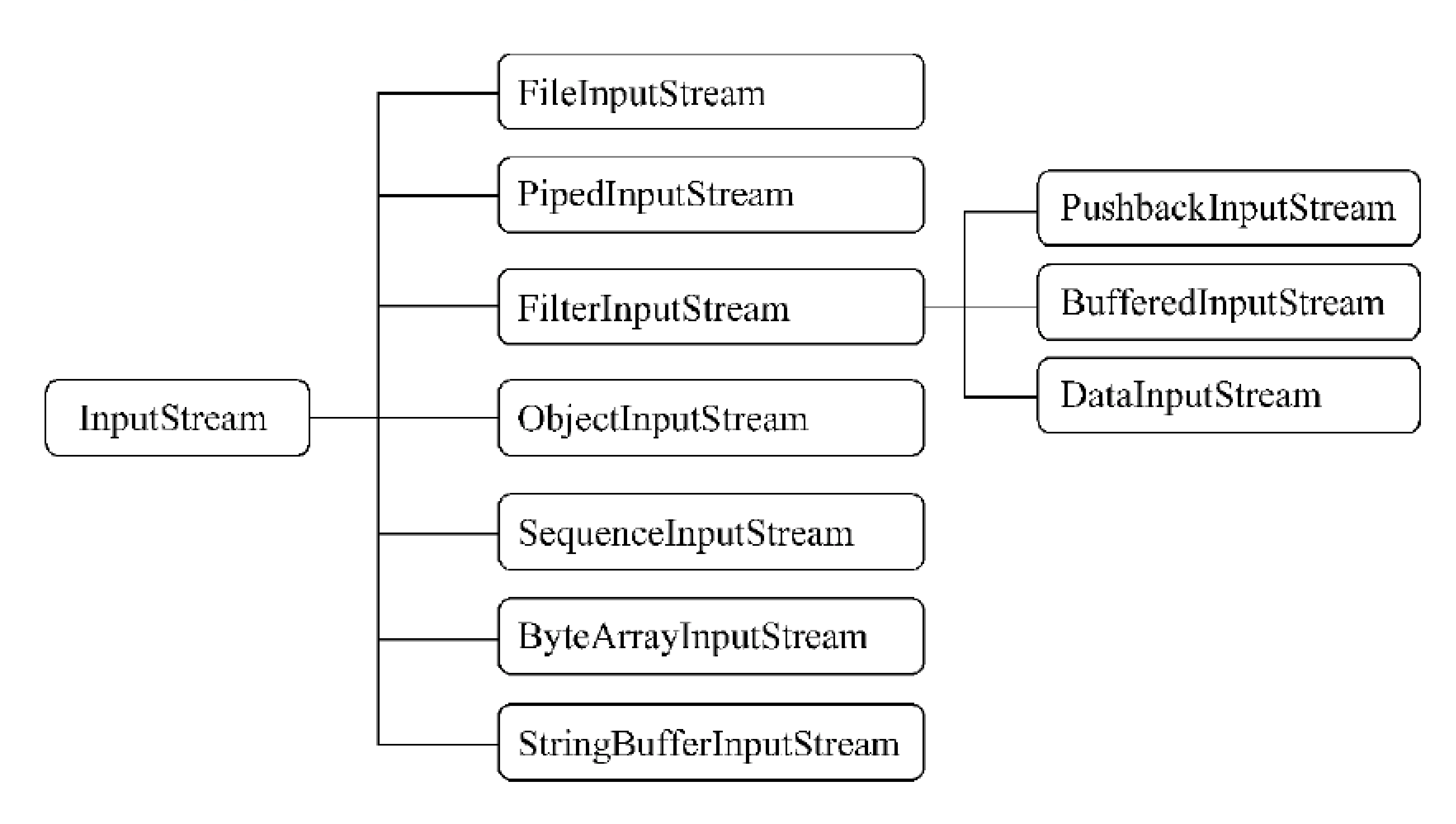


图 6-3　InputStream 类层次图

InputStream 类的常用方法有以下 5 个。

(1) int read()：输入流调用该方法从数据源中读取单个字节的数据，该方法返回字节值(0～255 之间的一个整数)。如果未读出字节就返回－1。

(2) int read(byte b[])：输入流调用该方法从数据源中试图读取 b. length 个字节到 b 中，返回实际读取的字节数目。如果到达文件的末尾，则返回－1。

(3) int read(byte b[],int off,int len)：输入流调用该方法从数据源中试图读取 len 个字节到 b 中，并返回实际读取的字节数目。如果到达文件的末尾，则返回－1。参数 off 指定从字节数组的某个位置开始存放读取的数据。

(4) void close()：输入流调用该方法关闭输入流。

(5) long skip(long numBytes)：输入流调用该方法跳过 numBytes 个字节，并返回实际跳过的字节数目。

2. OutputStream 类

OutputStream 类是所有字节输出流的超类，是定义了流式字节输出模式的抽象类，该类的所有方法返回一个 void 值并且在出错情况下引发一个 IOException 异常。

OutputStream 类层次图如图 6-4 所示。

OutputStream 类的常用方法如下。

(1) void write(int n)：输出流调用该方法向输出流写入单个字节。

(2) void write(byte b[])：输出流调用该方法向输出流写入一个字节数组。

(3) void write(byte b[],int off,int len)：从给定字节数组中，起始于偏移量 off 处取

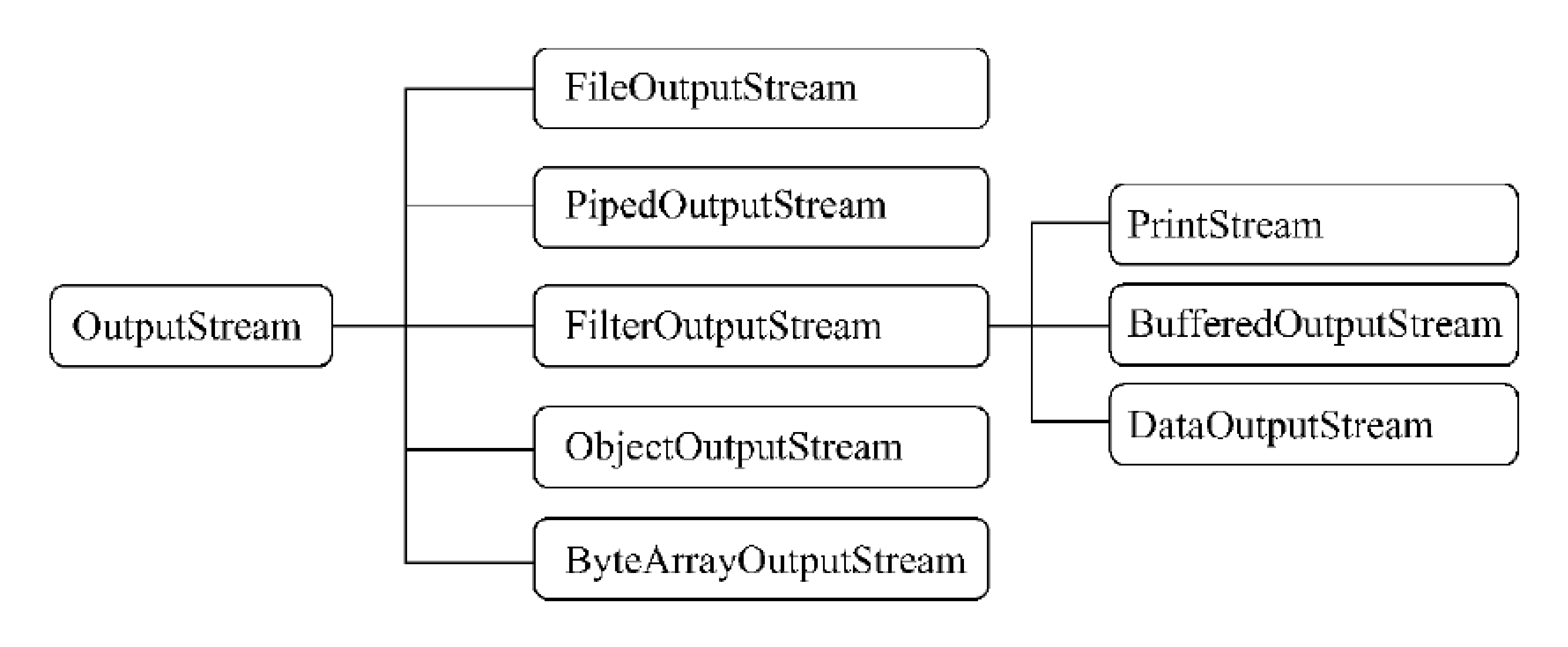


图 6-4 OutputStream 类层次图

len 个字节写到输出流。

（4）void close()：关闭输出流。

字节流有多个子类，能直接对文件进行读或写的子类有文件输入流（FileInputStream）和文件输出流（FileOutputStream），其中 FileInputStream 类可以文件名或 File 对象构造文件输入流对象，并通过文件输入流对象读取文件，它的构造方法如下。

- public FileInputStream(String name) throws FileNotFoundException
- public FileInputStream(File file) throws FileNotFoundException

FileOutputStream 类则可以文件名或 File 对象构造文件输出流对象，通过文件输出流对象写文件，它的构造方法如下。

- public FileOutputStream(String name) throws FileNotFoundException
- public FileOutputStream(File file) throws FileNotFoundException
- public FileOutputStream(String name, boolean append) throws FileNotFoundException

其中，name 为文件名，file 为文件类 File 对象，boolean append 表示文件的写入方式。其值为 false 时，为重写方式，即要写入的内容从文件开头写入，覆盖以前的文件内容；值为 true 时，为添加方式，即要写入的内容添加到文件的尾部。默认值是 false。

【例 6-2】 用类 FileInputStream 读文件。

本例以例 6-1 中创建的文件 newfile.txt 为参数构造 File 对象，再以 File 对象为参数构造输入流，循环读取输入流，并输出到客户端，本例源代码如下。本程序源代码存放于本书配套素材中的相应文件中（文件路径为 code\6\）。

```
<! -- 例程 6-2 readfile.jsp -->

<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.io.*" %>
<html>
  <body bgcolor=cyan>
    <font size=2>
      <%
          File f=new File("E:/code/6/test/newfile.txt");
          try
            {
              byte b[]=new byte[50]; //每次读取数据保存在该字节数组中
              int n=0;               //实际读取的字节数
              FileInputStream in=new FileInputStream(f);
```

```
                while((n = in.read(b))! =  - 1)
                    {
                    String temp = new String(b,0,n);
                    out.print(temp);
                  }
                in.close();
              }
            catch(IOException e)
              {
              }
        %>
      </font>
    </body>
  </html>
```

在本程序的代码中，使用输入流的 int read(byte b[])方法读取输入流，以文件名 newfile.txt 为参数，构造文件对象 f，再以文件对象 f 为参数，构造输入流 in，每次从 in 中读取 n 个字节，保存在数组 b 中，n 等于－1 时结束循环，否则，把数组 b 以字符串的方式输出到客户端。

程序的运行效果如图 6-5 所示。

图 6-5　readfile.jsp 的运行效果

【例 6-3】 用类 FileOutputStream 写文件。

本例创建一个表单用以接受客户端的文本输入，以文件名 write.txt 为参数创建一个输出流，把客户端输入的文本写入该输出流中，本例源代码如下。本程序源代码存放于本书配套素材中的相应文件中(文件路径为 code\6\)。

```
<! -- 例程 6 - 3 writefile.jsp -->

<% @ page contentType = "text/html;charset = GB2312"  %>
<% @ page import = "java.io. * " %>
<html>
  <body bgcolor = yellow>
  <font size = 2>
    <form action = "" method = post name = form>
        <INPUT type = "text" name = "boy">
        <INPUT TYPE = "submit" value = "保存" name = "submit">
    </form>
      <%
          String str = request.getParameter("boy");
          if(str == null) str = " ";
```

```
            try
            {
              byte buffer[ ] = str.getBytes("ISO - 8859 - 1");
              FileOutputStream wf = new FileOutputStream("E:/code/6/test/write.txt",true);
              wf.write(buffer); //将字节数组写入输出流指向的文件
              wf.close();       //关闭输出流
            out.println("将数据存入到文件 E:/code/6/test/write.txt 中");
          }
          catch(IOException ioe)
          {
            System.out.println("File Write Error!");
          }
        %>
      </body>
    </html>
```

在本程序的代码中，以 write.txt 为参数，创建一个输出流 wf，然后调用 write(byte b[])将 buffer 写入到输出流 wf 中，该程序运行后效果如图 6-6 所示。

在该页面中输入内容，如 aaa，并单击“保存”按钮，write.txt 文件中就会出现输入的内容，如图 6-7 所示。



图 6-6　writefile.jsp 的运行效果

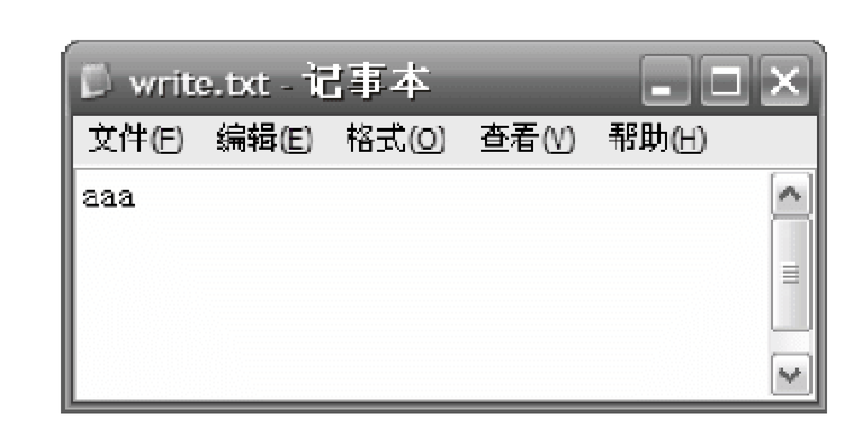


图 6-7　write.txt 文件

6.2.2　字符流

字节流可以读写文件，但字节流的处理单元为 1 个字节，对于占用两个字节以上的字符，如汉字(在文件中占用两个字节)，如果使用字节流读写文件会出现乱码现象，所以在 JSP 中提供了字符流。字符流处理的单元为两个字节的 Unicode 字符，提供了处理任何类型输入/输出操作的足够的功能，在 Unicode 字符集中，一个汉字被看作一个字符，采用字符流就可以避免乱码。

字符流有两个超类，也是两个抽象类，分别为字符输入流(Reader)和字符输出流(Writer)。

1．Reader 类

Reader 类是所有字符输入流的父类，也是定义 Java 的流式字符输入模式的抽象类，该类的所有方法在出错情况下都将引发 IOException 异常。

Reader 类层次图如图 6-8 所示。

Reader 的常用方法如下：

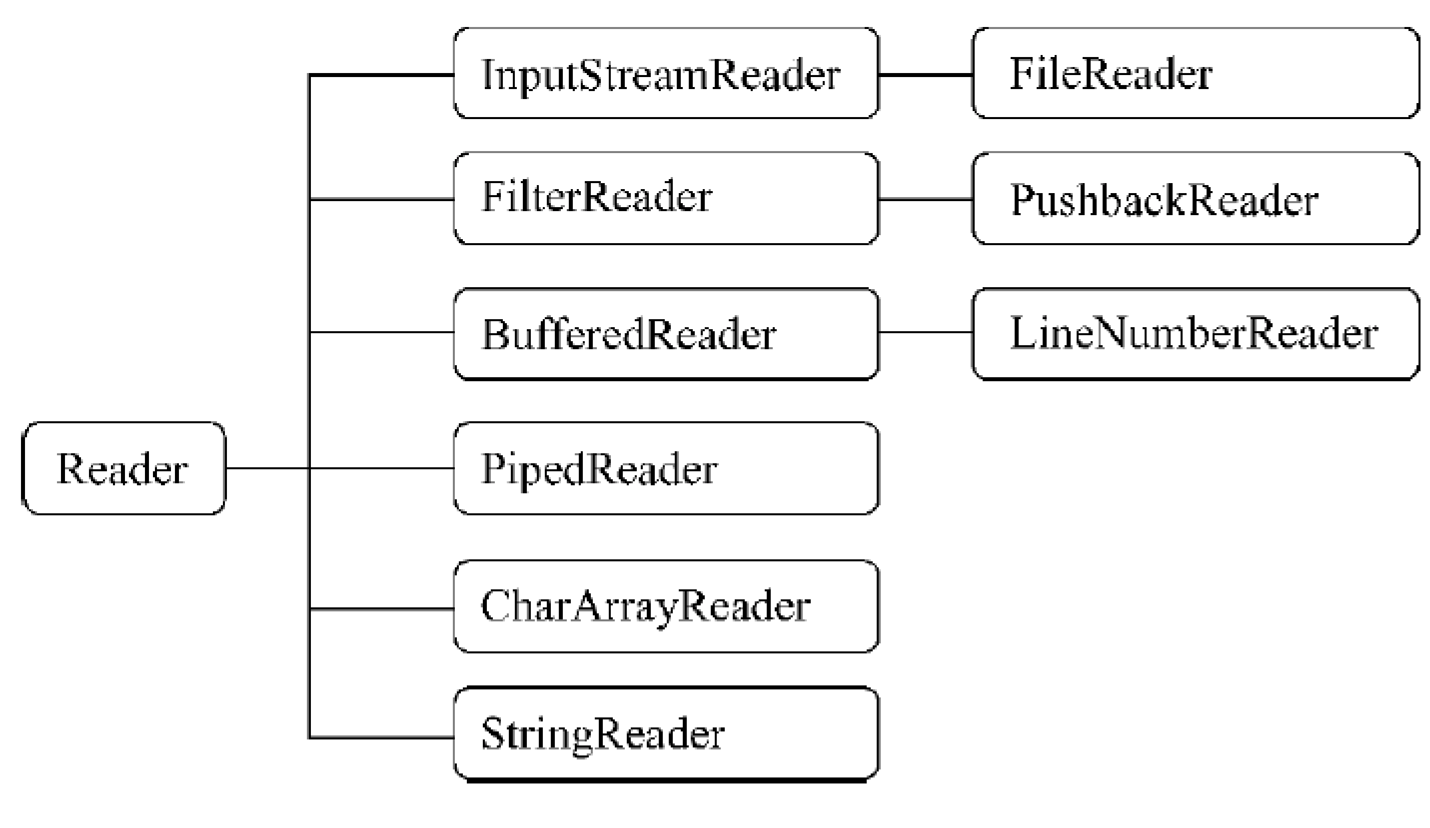


图 6-8　Reader 类层次图

(1) int read()：从源中读取一个字符，该方法返回一个参数(0～65535 之间的一个整数，Unicode 字符值)，如果未读出字符就返回－1。

(2) int read(char b[])：从源中读取 b. length 个字符到字符数组 b 中，返回实际读取的字符数目，如果到达文件的末尾，则返回－1。

(3) int read(char b[],int off,int n)：从源中读取 n 个字符并将其存放到字符数组 b 中，返回实际读取的字符数目。如果到达文件的末尾，则返回－1。其中 off 参数表明从数组 b 的 off 位移处开始存放数据。

(4) void close()：关闭输入流。

(5) long skip(long numBytes)：跳过 numBytes 个字符，并返回实际跳过的字符数目。

2. Writer 类

Writer 类是所有字符输出流的父类，是定义流式字符输出的抽象类，所有该类的方法都返回一个 void 值，并在出错条件下引发 IOException 异常。

Writer 类层次图如图 6-9 所示。

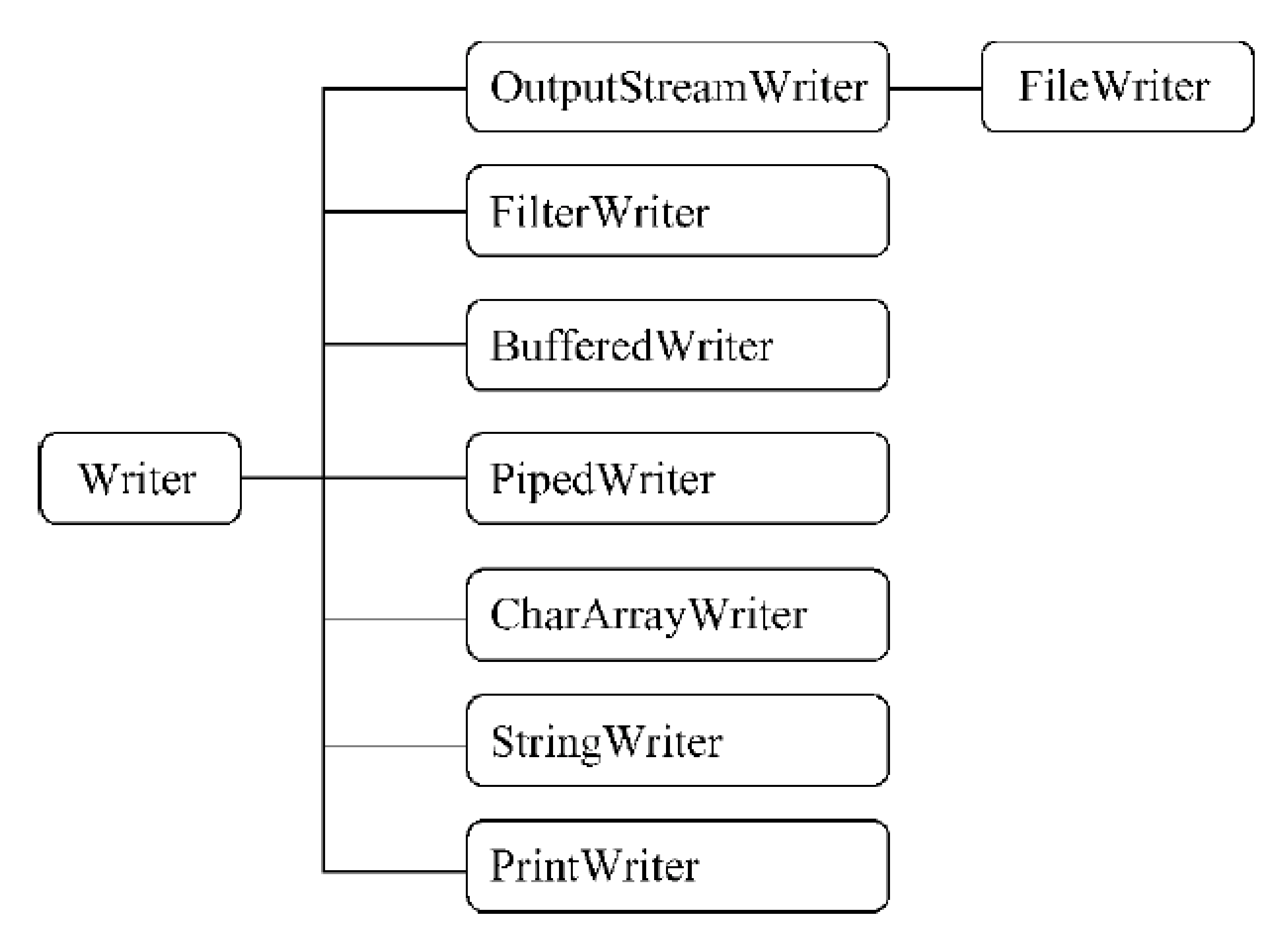


图 6-9　Writer 类层次图

Writer 的常用方法如下：

(1) void write(int n)：向输出流写入一个 Unicode 字符值(数字)。

(2) void write(char b[])：向输出流写入一个字符数组。

(3) void write(char b[],int off,int n):从给定字符数组中在偏移量 off 处取 n 个字符写到输出流。

(4) void write(String str):向输出流写入一个字符串。

(5) void close():关闭输出流。

字符流有多个子类,能直接对文件进行读或写的子类有文件字符输入流(FileReader)和文件字符输出流(FileWriter),其中 FileReader 类可以文件名或 File 对象构造文件输入流对象,并通过文件输入流对象读文件,它的构造方法如下。

- public FileReader(File file) throws IOException
- public FileReader(String name) throws IOException

FileWriter 类可以文件名或 File 对象构造文件输出流对象,通过文件输出流对象写文件,它的构造方法如下:

- public FileWriter(File file) throws IOException
- public FileWriter(String name) throws IOException
- public FileWriter(File file,boolean append) throws IOException
- public FileWriter(String name,boolean append) throws IOException

【例 6-4】 用类 FileReader 读文件。

本例是在客户端显示文件 newfile.txt 的内容。使用字符输入流的 int read(char b[])方法,读取输入流。本程序源代码存放于本书配套素材中的相应文件中(文件路径为 code\6\)。

程序源代码如下。

```
<! -- 例程 6 - 4 filereader.jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<%@ page import = "java.io. * " %>
<html>
  <body bgcolor = yellow>
    <font size = 2>
      <%
          File f = new File("E:/code/6/test/newfile.txt"); //构造文件对象 f
          try
            {
              FileReader in = new FileReader(f); //构造字符输入流 in
              String str = null;
              char b[ ] = new char[50];
              int n = 0;

      //每次从 in 中读取 n 个字符,保存在字符数组 b 中
              while((n = in.read(b))! = -1)
                {
                  str = new String(b,0,n);
                  out.print(str);
                }
              in.close();
          }
        catch(IOException e)
          {
```

```
            }
        %>
    </body>
</html>
```

程序以文件名 E:/code/6/test/newfile.txt 为参数构造文件对象 f，然后再以文件对象 f 为参数构造字符输入流 in，每次从 in 中读取 n 个字符，保存在字符数组 b 中，当 n 等于－1 时，结束循环，否则，把数组 b 以字符串的方式输出到客户端。该程序运行后的效果如图 6-10 所示。

古老的礼赞来山东出差，公干之余提出去曲阜拜孔之。车子还没有进曲阜城，人就有一种对古老的崇敬，在那一带，空气中似乎总蕴涵着悠久的气息。这并非全部来自孔庙的大殿、阙里的牌坊、街心的钟楼、路旁的古柏，仅仅曲阜这两个字，就已经印证着或代表了悠久和古老。不过现在的曲阜难以形容，先哲的呼吸仍然飘忽在这敦厚却有些陈旧的小城中，未来的气息却弥漫并已经使古老的小城有点浮躁。我不知是应该感慨古老，还是迷惑于现代。在颜庙大家谈起颜回，颇有些感慨，简而言之：儒家之所以能够被封建统治阶级接受，或许就是因为颜回的甘于清贫。颜回是孔子最得意的弟子，是“四配”之一。《论语》中有一段称赞颜回的：子曰：“贤哉，回也！一箪食，一瓢饮，在陋巷，人不堪其忧，回也不该起乐。贤哉，回也！”凭这几句话，几千年来，颜回成为中国人做人为民的道德规范 。可惜的是，颜回只活了41年。据说颜回死?螅?涓盖肭罂子勇粝陆底游?颜回做外椁?可孔子不同意：“我曾做过大夫，是不可以步行的。”孔子的儿子孔俚死后也同样没有外椁。这就是“克己复礼”。颜回地下没有外椁，人们却为他在地上修建了巍峨的颜庙。更重要的是，人们因此记住了颜回。

图 6-10　fileReader.jsp 的运行效果

【例 6-5】 用类 FileWriter 写文件。

本例以文件名 write.txt 为参数构造输出流，调用 write(char b[])方法，向输出流写字符数组，把从客户端输入的文本添加到服务器的 write.txt 文件中。本程序源代码存放于本书配套素材中的相应文件中(文件路径为 code\6\)。

本程序源代码如下。

```
<! -- 例程 6 - 5 filewriter.jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<%@ page import = "java.io.*" %>
<html>
 <body bgcolor = yellow>
  <font size = 3>
   <form action = "" method = post name = form>
     <input type = "text" name = "boy">
     <input type = "submit" value = "保存" name = "submit">
   </form>
    <%
     String str = request.getParameter("boy");
     if(str == null)
        str = " ";
     byte b[ ] = str.getBytes("ISO - 8859-1");
     str = new String(b);
     try
      {
         //以"E:/code/6/test/write.txt"为参数，创建一个输出流 wf
        FileWriter wf = new FileWriter("E:/code/6/test/write.txt",false);
        wf.write(str);          //将字符串写入输出流指向的文件
        wf.close();
        out.println("将数据存入到文件:e:/write.txt 中");
```

```
        }
        catch(IOException ioe)
        {
            System.out.println("File Write Error!");
        }
    %>
  </body>
</html>
```

该程序运行后的效果与例6-3一样。

6.2.3 数据流

前面的字符流只能对文件进行字符类型的数据读写，字节流只能对文件进行字节类型的数据读写，只有数据流才能对文件进行各种数据类型(Sava基本类型数据)的数据读写。

数据流包括数据输入流(DataInputStream)和数据输出流(DataOutputStream)。

1. 数据流的构造方法

(1) 数据输入流的构造方法：public DataInputStream(InputStream in)。

(2) 数据输出流的构造方法：public DataOutputStream(OutputStream out)。

其中，in是输入流对象，out是输出流对象。

2. 数据流的常用方法

1) 数据输入流常用方法

(1) close()：关闭流。

(2) readBoolean()：读取一个布尔值。

(3) readByte()：读取一个字节。

(4) readChar()：读取一个字符。

(5) readDouble()：读取一个双精度浮点值。

(6) readFloat()：读取一个单精度浮点值。

(7) readInt()：从文件中读取一个int值。

(8) readLong()：读取一个长型值。

(9) readShort()：读取一个短型值。

(10) readUnsignedByte()：读取一个无符号字节。

(11) readUnsignedShort()：读取一个无符号短型值。

(12) readUTF()：读取一个UTF字符串。

2) 数据输出流常用方法

(1) close()：关闭流。

(2) writeBoolean(boolean v)：把一个布尔值作为单字节值写入。

(3) writeBytes(String s)：写入一个字符串。

(4) writeChar(String s)：写入字符串。

(5) writeDouble(double v)：写入一个双精度浮点值。

(6) writeFloat(float v)：写入一个单精度浮点值。

(7) writeInt(int v)：写入一个 int 值。

(8) writeLong(long v)：写入一个长型值。

(9) writeShort(int v)：写入一个短型值。

(10) writeUTF(String s)：写入一个 UTF 字符串。

3. 流链

在实际应用中，利用各种流的特点，将多个流套接在一起可构成一个流链。程序通过输入流链读取数据源点数据，通过输出流链向数据终点写数据。这里的数据源点和数据终点一般指文件或内存。下面介绍输入流管道模型和输出流管道模型。

1) 输入流链

输入流管道模型如图 6-11 所示。

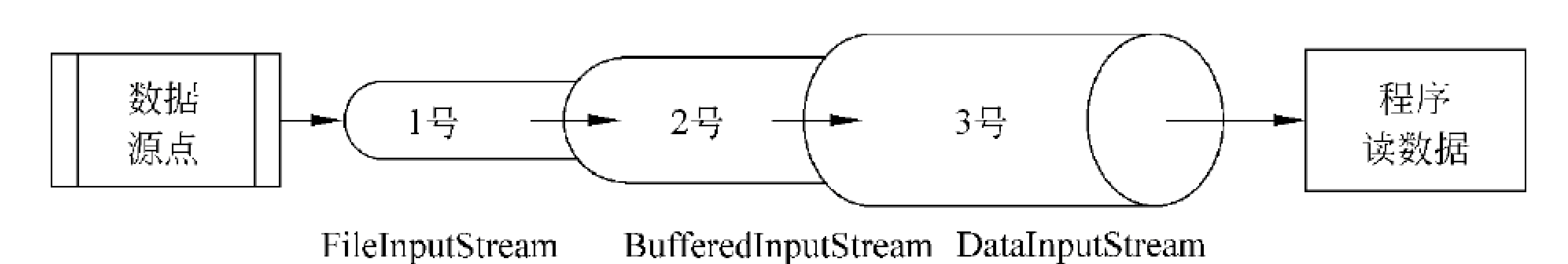


图 6-11　输入流管道模型

输入管道有 3 种型号，分别为 1 号(FileInputStream)、2 号(BufferedInputStream)和 3 号(DataInputStream)，每种管道代表一种流，将它们进行管道套接后，可以组成 4 种输入流链，用户可以选择其中的任意一种流链，从数据源点读取数据。这 4 种输入流链如下。

第 1 种流链：仅由 1 号构成的流，程序通过 FileInputStream 对象读数据。

第 2 种流链：由 1 号和 2 号套接构成的流，程序通过 BufferedInputStream 对象读数据。

第 3 种流链：由 1 号、2 号和 3 号套接构成的流，程序通过 DataInputStream 对象读取数据。

第 4 种流链：由 1 号和 3 号套接构成的流，程序通过 DataInputStream 对象读数据。

2) 输出流链

输出流管道模型如图 6-12 所示。

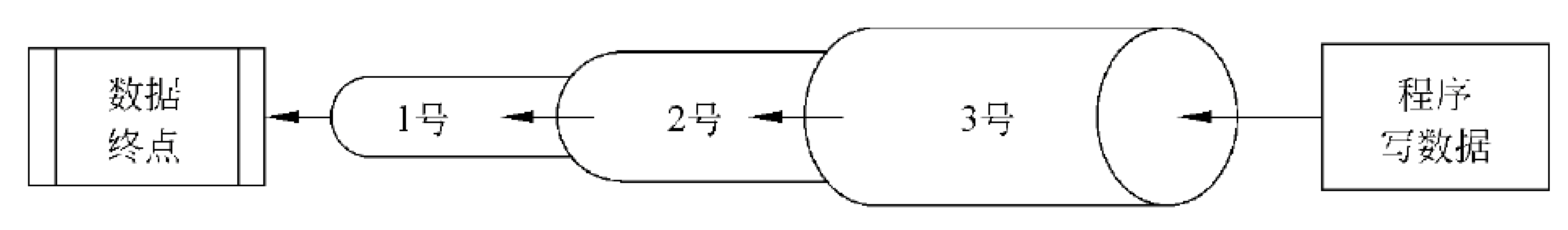


图 6-12　输出流管道模型

输出管道也有 3 种型号，分别为 1 号(FileOutputStream)、2 号(BufferedOutputStream)和 3 号(DataOutputStream)，每种管道代表一种流，将它们进行管道套接后，同样可以组成 4 种输出流链，用户可以选择其中的任意一种流链，向数据终点写入数据。这 4 种输出流链如下。

第 1 种流链：仅由 1 号构成的流，程序通过 FileOutputStream 对象向数据终点写数据。

第 2 种流链：由 1 号和 2 号套接构成的流，程序通过 BufferedOutputStream 对象向数据终点写数据。

第 3 种流链：由 1 号、2 号和 3 号套接构成的流，程序通过 DataOutputStream 对象向数据终点写数据。

第 4 种流链：由 1 号和 3 号套接构成的流，程序通过 DataOutputStream 对象向数据终点写数据。

【例 6-6】 使用数据流实现录入成绩单和显示成绩单。

本例使用两个页面完成此功能，由一个页面 cjlulu. jsp 提供录入界面，并把成绩保存到文本文件中；再由另一页面 showresult. jsp 读取文本文件中的数据，显示到客户端。本程序源代码存放于本书配套素材中的相应文件中（文件路径为 code\6\）。

cjlulu. jsp 程序的源代码如下：

```
                    <! -- 例程 6 - 6 cjlulu.jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<%@ page import = "java.io.*" %>
<html>
<body>
<P> <center>在下面的表格输入成绩：</center>
   <FORM action = "" method = post name = form>
   <Table align = "CENTER" Border = 1 >
      <TR>
          <TH width = 40 > 姓名</TH>
          <TH width = 40 > 数学</TH>
          <TH width = 40 > 英语</TH>
          <TH width = 80 > 计算机</TH>
      </TR>
   <%
      int i = 0;
      while(i <= 3)
        {
          out.print("<TR>");
          out.print("<TD>");
          out.print("<INPUT type = text name = name value = >");
          out.print("</TD>");
          out.print("<TD>");
          out.print("<INPUT type = text name = math value = 0 >");
          out.print("</TD>");
          out.print("<TD>");
          out.print("<INPUT type = text name = english value = 0 >");
          out.print("</TD>");
          out.print("<TD>");
          out.print("<INPUT type = text name = computer value = 0 >");
          out.print("</TD>");
          out.print("</TR>");
          i++;
        }
```

```
        %>
          <TR>
              <td>
              </td>
              <td  align=center>
              <INPUT type=submit name="g" value="保存成绩">
              </td>
          </TR>
      </Table>
    </form>
      <%
        String name[] = request.getParameterValues("name");
        String math[] = request.getParameterValues("math");
        String english[] = request.getParameterValues("english");
        String computer[] = request.getParameterValues("computer");
        try
          {
            File f = new File("E:/code/6/test/student.txt");
            FileOutputStream o = new FileOutputStream(f,true);
            DataOutputStream DataOut = new DataOutputStream(o);
            for(int k = 0;k<name.length;k++)
              {
                DataOut.writeUTF(name[k]);
                DataOut.writeUTF(math[k]);
                DataOut.writeUTF(english[k]);
                DataOut.writeUTF(computer[k]);
              }
            DataOut.close();
            o.close();
          }
        catch(IOException e)
          {
          }
        catch(NullPointerException ee)
          {
          }
      %>
      <center>
        <A href=showresult.jsp><BR> 查看成绩>
      </center>
    </body>
  </html>
```

该程序的运行效果如图 6-13 所示。

该页面提供了一个成绩录入的表单窗口，程序从该表单中获取成绩数据，以 E:/code/6/test/student.txt 为参数，创建 File 对象 f，然后对 f 进行两次封装，得到数据输出流 DataOut，最后将成绩写入输出流 DataOut。

在该页面中录入成绩后，单击“保存成绩”按钮，程序则调用 writeUTF(String str)方法向文件中写入 UTF 格式的字符串。

在下面的表格输入成绩：

姓名	数学	英语	计算机
赵海丰	75	73	82
张琳	85	86	90
刘小英	68	75	85
林茵	90	78	76
	保存成绩		

查看成绩>

图 6-13 cjlulu. jsp 程序的运行效果

单击“查看成绩”按钮后，页面转到 showresult. jsp 页面，读取 E:/code/6/test/student. txt 文件中的数据，然后输出到客户端。showresult. jsp 程序的源代码如下。

```
<! -- 例程 6 - 7 showresult. jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<%@ page import = "java. io. * " %>
<html>
 <body>
  <center>
   <P><b> 成绩单 </b>
    <%
     try
     {
        File f = new File("E:/code/6/test/student. txt");
        FileInputStream in = new FileInputStream(f);
        DataInputStream DataIn = new DataInputStream(in);
        String name = "ok";
        String math = "0",english = "0",computer = "0";
        out. print("<Table  Border = 1 >");
        out. print("<TR>");
        out. print("<TH width = 50 > 姓名</TH>");
        out. print("<TH width = 50 > 数学</TH>");
        out. print("<TH width = 50 > 英语</TH>");
        out. print("<TH width = 100 > 计算机</TH>");
        out. print("</TR>");
        while((name = DataIn. readUTF())! = null)
          {
            byte bb[ ] = name. getBytes("ISO - 8859 - 1");
            name = new String(bb);
            math = DataIn. readUTF();
            english = DataIn. readUTF();
            computer = DataIn. readUTF();
            out. print("< TR >");
            out. print("< TD width = 100 >");
            out. print(name);
            out. print("</TD >");
            out. print("< TD >");
            out. print(math);
```

```
                out.print("</TD>");
                out.print("<TD>");
                out.print(english);
                out.print("</TD>");
                out.print("<TD>");
                out.print(computer);
                out.print("</TD>");
                out.print("</TR>");
              }
            out.print("</Table>");
            DataIn.close();
            in.close();
          }
      catch(IOException ee)
        {   }
      %>
    </center>
    </BODY>
    </HTML>
```

该程序仍以 E:/code/6/test/student.txt 为参数，创建 File 对象 f，然后对 f 进行两次封装，创建输入流对象 DataIn，调用 readUTF()方法，读取一个 UTF 字符串。其运行效果如图 6-14 所示。

成绩单

姓名	数学	英语	计算机
赵海丰	75	73	82
张琳	85	86	90
刘小英	68	75	85
林茵	90	78	76

图 6-14 showresult.jsp 的运行效果

6.2.4 对象流

使用对象流可以直接把对象写入文件，也可以直接从文件中读取一个对象。对象流分为对象输入流(ObjectInputStream)和对象输出流(ObjectOutputStream)。

1. 对象流的构造方法

(1) 对象输入流的构造方法：public objectInputStream(InputStream in) throws IOException。

(2) 对象输出流的构造方法：public ObjectOutputStream(OutputStream out) throws IOException。

可见，要用对象流对文件进行读写，必须对文件进行两次构造。

2. 对象流的实例方法

(1) 对象输入流的实例方法：public final Object readObject() throws OptionalDataException, ClassNotFoundException, IOException。

(2) 对象输出流的实例方法：public final void writeObject(Object obj) throws IOException。

【例 6-7】 使用对象流实现货物(货物名称、货物数量)的录入、删除和显示。

本例由 4 个 JSP 页面构成，goods.jsp 页面提供数据录入界面，并把数据提交给 input.

jsp 页面；input.jsp 页面把数据保存到 E:\code\6\test\goods.txt 文件中；del.jsp 页面以货物名为关键字删除 hashtable 对象中的相应数据；showgoods.jsp 页面显示 hashtable 对象中的所有数据。4 个页面的交互关系如图 6-15 所示。本程序源代码存放于本书配套素材中的相应文件中（文件路径为 code\6\）。

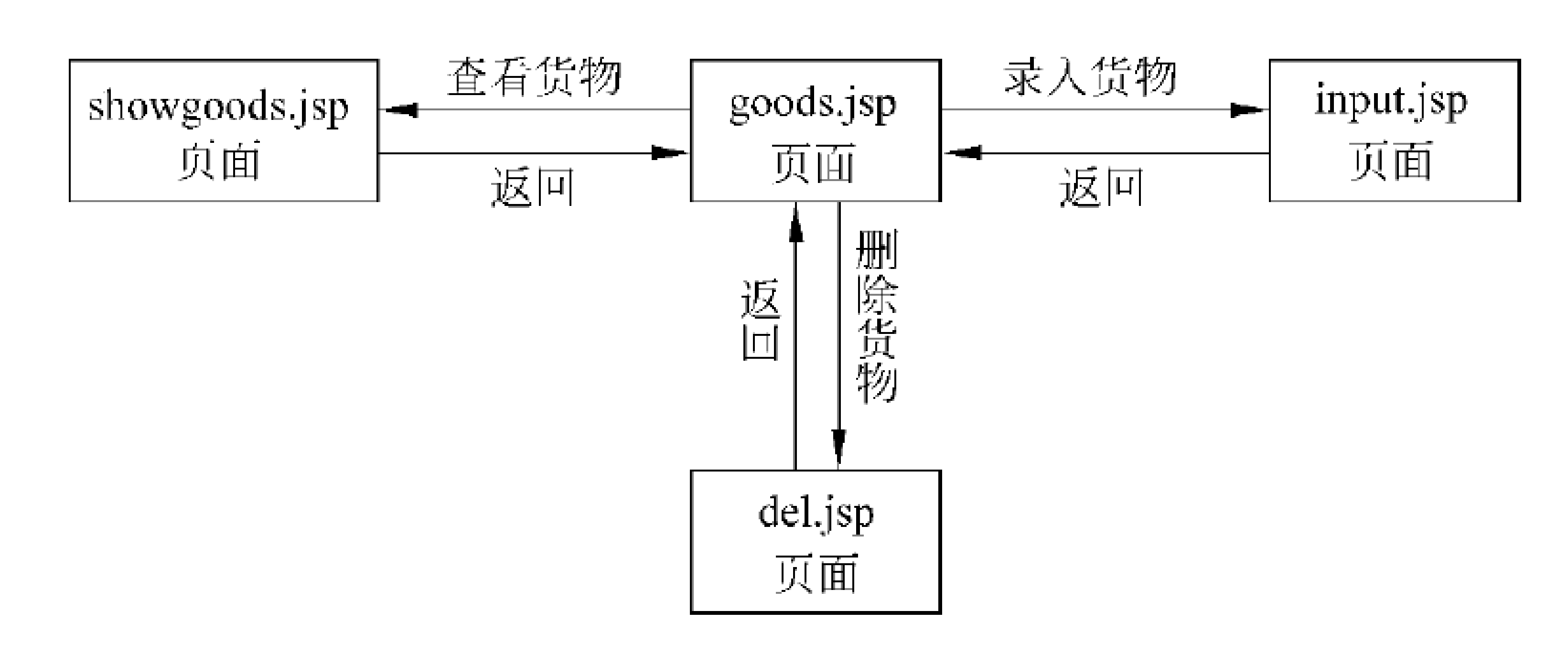


图 6-15　页面交互式关系

goods.jsp 页面的程序源代码如下。

```
<!-- 例程 6-8 goods.jsp -->

<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import ="java.io.*" %>
<%@ page import ="java.util.*" %>
<HTML>
<BODY>
  <center>
  <P><b>货物录入界面</b>
  <FORM action="input.jsp" method=post>
    <P>货物名称：
    <INPUT type=text name="N">
    <P>货物数量：
    <INPUT type=text name="M">
    <BR>
    <INPUT type=submit value="录入货物">
   </FORM>
  <Table Border=0   height=5>
    <tr>
     <td>
        <FORM action="showgoods.jsp" method=post>
          <INPUT type=submit value="查看货物">
        </FORM>
     </td>
     <td>
        <FORM action="del.jsp" method=post>
          <INPUT type=submit value="删除货物">
        </FORM>
     </td>
    </tr>
  </table>
 </center>
```

```
</BODY>
</HTML>
```

该程序主要创建一个表单，用于录入货物名称和货物数量，程序运行后的显示效果如图 6-16 所示。



图 6-16 goods.jsp 的运行效果

在该页面中输入货物名称和货物数量后，单击“录入货物”按钮，程序把数据提交给 input.jsp 页面，该页面的程序源代码如下。

<! -- 例程 6 - 9 input.jsp -->

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import ="java.io.*" %>
<%@ page import ="java.util.*" %>
<HTML>
 <BODY bgcolor=yellow>
  <%!
         Hashtable hashtable=new Hashtable(); //创建一个 Hashtable 对象
         synchronized void putGoodsToHashtable(String key,String list)
         {
            hashtable.put(key,list);
         }
  %>
  <%
       String name=(String)request.getParameter("N");
       String mount=(String)request.getParameter("M");
       name=name.trim();
       byte c[]=name.getBytes("ISO-8859-1");
       name=new String(c);
       byte d[]=mount.getBytes("ISO-8859-1");
       mount=new String(d);
       File f=new File("E:/code/6/test/goods.txt");
       if(f.exists())
       {
        try
        {
          FileInputStream in=new FileInputStream(f);
           ObjectInputStream object_in=new ObjectInputStream(in);
          hashtable=(Hashtable)object_in.readObject();
          object_in.close();
          in.close();
          if(hashtable.containsKey(name))
          {
            session.setAttribute("name",name);
            response.sendRedirect("del.jsp"); //重定向到删除页面
           }
          else
           {
            String s="#"+name+"#"+mount+"#";
            putGoodsToHashtable(name,s);
          try
```

```
        {
          FileOutputStream o = new FileOutputStream(f);
          ObjectOutputStream object_out = new ObjectOutputStream(o);
          object_out.writeObject(hashtable);
          object_out.close();
          o.close();
         }
        catch(Exception eee) { }
        out.print("<BR>" + "您已经将货物<b>[" + name + "]</b>存入文件中的 hashtable 表中");
     }
    }
   catch(IOException e) { }
   }
 else
 {
  String s = "#" + name + "#" + mount + "#";
  putGoodsToHashtable(name,s);
 try
 {
   FileOutputStream o = new FileOutputStream(f);
   ObjectOutputStream object_out = new ObjectOutputStream(o);
   object_out.writeObject(hashtable);
   object_out.close();
   o.close();
   out.print("<BR>" + "您是第一个录入货物的人");
   out.print("<BR>" + "货物的名称是<b>[" + name + "]</b>");
  }
 catch(Exception eee) { }
}
%>
<FORM action = "goods.jsp"  method = post >
    <INPUT type = submit value = "返回">
</FORM>
</BODY>
</HTML>
```

【说明】

(1) 在程序中首先创建一个 hashtable 对象，将每次录入的数据保存在 Hashtable 类型的对象中。

(2) 从 goods.jsp 页面的表单中获取货物名称和数量，以 E:\code\6\test\goods.txt 为参数，创建文件对象 f(该文件用于保存 hashtable 对象)。

(3) 判断文件 E:\code\6\test\goods.txt 是否存在，若不存在，则用"#"+name+"#"+mount+"#"构造字符串 s，把(name,s)添加到 hashtable 中，再把 hashtable 写入文件 E:\code\6\test\goods.txt 中，其页面的显示效果如图 6-17 所示。

(4) 从文件 E:\code\6\test\goods.txt 中获取 hashtable 对象，若 hashtable 中已存在名称为 name 的货物，则用"name"标示属性名，用 name 标示属性值，将该属性名-值对加入到 session 对象中，然后转向 del.jsp 页面。否则，用"#"+name+"#"+mount+"#"构造字符串 s，把(name,s)添加到 hashtable 中，再把 hashtable 写入文件中，其页面显示效果如

图 6-18 所示。

您是第一个录入货物的人
货物的名称是 [电视机]
返回

图 6-17　input.jsp 运行效果

您已经将货物 [电视机] 存入文件中的hashtable表中
返回

图 6-18　获取对象后的运行效果

del.jsp 页面实现删除功能，其程序源代码如下。

```
<!-- 例程 6-10 del.jsp -->

<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import ="java.io.*" %>
<%@ page import ="java.util.*" %>
<HTML>
<BODY>
  <center>
  <%!
     Hashtable hashtable=new Hashtable();
     synchronized void removeGoodsToHashtable(String key)
     {
        hashtable.remove(key);
     }
     synchronized void putGoodsToHashtable(String key,String list)
     {
        hashtable.put(key,list);
     }
   %>
  <%
     String oldname=(String)session.getAttribute("name");
     out.print("<BR><b>货物删除界面</b><br>");
     out.print("<FORM action=del.jsp method=get");
     out.print("<P>要删除的货物名称：");
     out.print("<INPUT type=text name=N value="+oldname+">");
     out.print("<INPUT type=submit name=del value=del>");
     out.print("<br>");
     out.print("<br>");
     out.print("<br>");
     out.print("</FORM>");
     String name=request.getParameter("N");
     if(name==null) name="";
     name=name.trim();
     byte c[]=name.getBytes("ISO-8859-1");
     name=new String(c);
     try
        {
           File f=new File("E:/code/6/test/goods.txt");
           FileInputStream in=new FileInputStream(f);
           ObjectInputStream object_in=new ObjectInputStream(in);
           hashtable=(Hashtable)object_in.readObject();
```

```
            object_in.close();
            in.close();
            if(hashtable.containsKey(name))
               {
                  removeGoodsToHashtable(name);
                  out.print("<BR>" + "货号：" + name + "的信息被删除");
                  FileOutputStream o = new FileOutputStream(f);
                  ObjectOutputStream object_out = new ObjectOutputStream(o);
                  object_out.writeObject(hashtable);
                  object_out.close();
                  o.close();
               }
             else
               {
                  out.print("<BR>" + "没有货号：" + name + "记录");
               }
         }
       catch(Exception ee) { }
      %>
   <FORM action = "goods.jsp" method = post >
       <INPUT type = submit value = "返回">
   </FORM>
  </center>
</BODY>
</HTML>
```

该程序运行后的显示效果如图 6-19 所示。

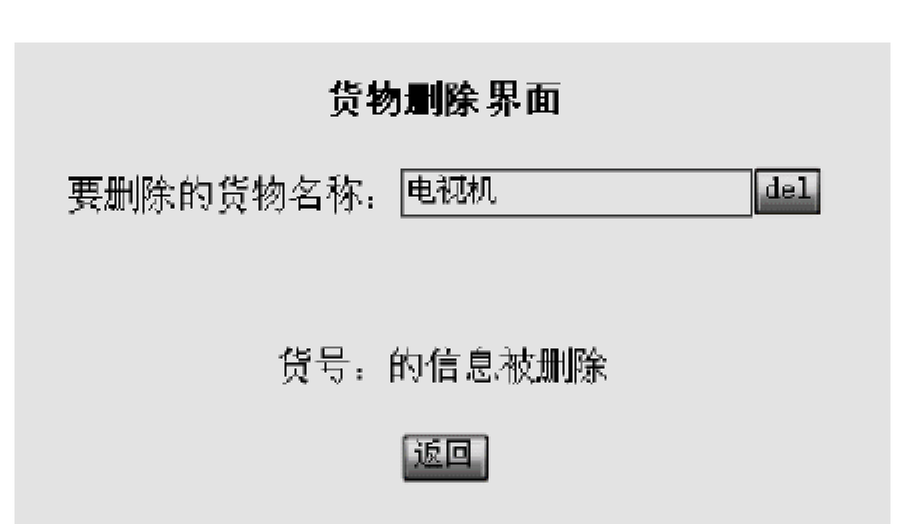


图 6-19 del.jsp 页面的运行效果

查看货物页面 showgoods.jsp。从 E:/goods.txt 文件中获取 hashtable 对象，并输出到客户端。该程序的源代码如下。

```
<!-- 例程 6-11 showgoods.jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<%@ page import  = "java.io.*" %>
<%@ page import  = "java.util.*" %>
<HTML>
<BODY>
<center>
<P><b>货物查询界面</b><br><br>
  <%
    try
      {
        File f = new File("E:/code/6/test/goods.txt");
        FileInputStream in = new FileInputStream(f);
        ObjectInputStream object_in = new ObjectInputStream(in);
        Hashtable hashtable = (Hashtable)object_in.readObject();
        object_in.close();
        in.close();
        Enumeration enum = hashtable.elements();
        out.print("<table border = 1 >");
```

```
            out.print("<tr>");
            out.print("<th bagcolor = yellow>" + "货物名称" + "</th>");
            out.print("<th bagcolor = yellow>" + "货物数量" + "</th>");
            out.print("</tr>");
            while(enum.hasMoreElements())
              {
                out.print("<tr>");
                String goods = (String)enum.nextElement();
                StringTokenizer fenxi = new StringTokenizer(goods,"#");
                while(fenxi.hasMoreTokens())
                  {
                    String str = fenxi.nextToken();
                    out.print("<td bagcolor = yellow>" + str + "</td>");
                  }
                out.print("</tr>");
              }
            out.print("</table>");
            hashtable.clear();
          }
        catch(Exception event)
          {
            out.println("无法读出");
          }
      %>
      <FORM action = "goods.jsp" method = post>
          <INPUT type = submit value = "返回">
      </FORM>
    </center>
    </BODY>
    </HTML>
```

6.3 随机访问类

前面几节介绍的数据流只能按顺序读写文件,而且输入流只能读不能写,输出流只能写不能读,即不能使用同一个流对文件进行读写操作。为了解决这个问题,JSP 提供了随机访问类 RandomAccessFile,使用该类对象可以随机读写文件。本节将介绍该类的构造方法和实例方法。

6.3.1 构造方法

随机访问类 RandomAccessFile 的构造方法如下:

- public RandomAccessFile(File file,String mode)throws FileNotFoundException
- public RandomAccessFile(String name,String mode)throws FileNotFoundException

其中,name 表示文件名;file 表示文件对象;mode 指定对文件的访问模式;r 表示读; w 表示写; rw 表示读/写。当文件不存在时,构造方法将抛出 FileNotFoundException 异常。

6.3.2 实例方法

随机访问类 RandomAccessFile 的常用方法如表 6-1 所示。

表 6-1　RandomAccessFile 类的常用方法

常用方法	说明
close()	关闭文件
getFilePointer()	获取文件指针的位置
Length()	获取文件的长度
read()	从文件读取一个字节的数据
readBoolean()	从文件中读取一个布尔值,0 代表 flase,其他代表 true
readByte()	从文件中读取一个字节
readChar()	从文件中读取一个字符(2 个字节)
readDouble()	从文件中读取一个双精度浮点值(8 个字节)
readFloat()	从文件中读取一个单精度浮点值(4 个字节)
readFully(byte b[])	读 b.length 字节放到数组 b,完全填满该数组
readInt()	从文件中读取一个 int 值(4 个字节)
readLine()	从文件中读取一个文本行
readLong()	从文件中读取一个长型值(8 个字节)
readShort()	从文件中读取一个短型值(2 个字节)
readUTF()	从文件中读取一个 UTF 字符串
seek()	定位文件指针在文件中的位置
setLength(long newlength)	设置文件的长度
skipByte(int n)	从文件中跳过给定数量的字节
Write(byte b[])	写 b.length 个字节到文件
writeBoolean(boolean v)	把一个布尔值作为单字节值写入文件
writeByte(int v)	向文件写入一个字节
writeBytes(String s)	向文件写入一个字符串
writeChar(char c)	向文件写入一个字符
writeChars(String s)	向文件写入一个作为字符数据的字符串
writeDouble(double v)	向文件写入一个双精度浮点值
writeFloat(float v)	向文件写入一个单精度浮点值
writeInt(int v)	向文件写入一个 int 值
writeLong(long v)	向文件写入一个长型 int 值
writeShort(int v)	向文件写入一个短型 int 值
writeUTF(String s)	写入一个 UTF 字符串

以上方法出错时将抛出 IOException 异常,当读到文件尾时将抛出 EOFException 异常。

【例 6-8】 使用 RandomAccessFile 类定位文件。

在 E:\code\6\test 目录下,有四部小说。本例在网上提供一个窗口,可以让所有客户选择其中一部小说继续写作。本例用 3 个页面来实现,xuanze.jsp 页面提供客户选择小说的界面,在该页面中把客户选择的小说名提交给 write.jsp 页面;write.jsp 页面则提供续写

小说的界面，并把续写的内容提交给 save.jsp 页面；save.jsp 页面把小说的内容保存到 E:\code\6\test 目录下相应的文件中。本程序源代码存放于本书配套素材中的相应文件中（文件路径为 code\6\）。

xuanze.jsp 页面的程序源代码如下。

```
<! -- 例程 6 - 12 xuanze.jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<%@ page import = "java.io.*" %>
<HTML>
<BODY bgcolor = cyan><FONT size = 2>
  <%
    String str = response.encodeURL("write.jsp");
  %>
  <P>选择您想续写小说的名字:<BR>
  <FORM action = "<% = str %>" method = post name = form>
     <BR><INPUT type = "radio" name = "R" value = "hlm.doc" >红楼梦
     <BR><INPUT type = "radio" name = "R" value = "xyj.doc" >西游记
     <BR><INPUT type = "radio" name = "R" value = "shz.doc" >水浒传
     <BR><INPUT type = "radio" name = "R" value = "sgyy.doc" >三国演义
     <BR><INPUT type = submit name = "g" value = "提交">
  </FORM>
</FONT>
</BODY>
</HTML>
```

该程序运行后的效果如图 6-20 所示。

图 6-20　xuanze.jsp 的运行效果

在该页面中选择要续写的小说名后，单击“提交”按钮，页面转到 write.jsp 页面，该页面程序的源代码如下：

```
<! -- 例程 6 - 13 write.jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<%@ page import = "java.io.*" %>
<HTML>
<BODY bgcolor = cyan><FONT size = 2>
  <P>小说已有内容:
  <Font size = 2 Color = Navy>
  <%
     String str = response.encodeURL("save.jsp");
  %>
  <% - - 获取客户提交的小说的名字 - - %>
  <%
     String name = (String)request.getParameter("R");
     if(name == null)
       {
         name = "ee";
       }
     byte c[] = name.getBytes("ISO - 8859 - 1");
     name = new String(c);
     session.setAttribute("name",name);
     File storyFileDir = new File("E:/code/6","test"); //创建一个代表目录的 File 对象
```

```
        storyFileDir.mkdir(); //创建一个目录 story
        File f = new File(storyFileDir,name);
        //列出小说的内容
        try
          {
             RandomAccessFile file = new RandomAccessFile(f,"r");
             String temp = null;
             while((temp = file.readUTF())! = null)
                {
                   byte d[ ] = temp.getBytes("ISO - 8859 - 1");
                   temp = new String(d);
                   out.print("< BR >" + temp);
                }
             file.close();
          }
        catch(IOException e) { }
      % >
     </FONT >
     < P >请输入续写的内容:
     < Form action = "< %  = str % >" method = post name = form >
        < TEXTAREA name = "messages" ROWs = "12" COLS = 80 WRAP = "physical">
        </TEXTAREA >
        < BR >
        < INPUT type = "submit" value = "提交信息" name = "submit">
     </FORM >
   </BODY >
   </HTML >
```

该程序运行后的效果如图 6-21 所示。



图 6-21　write.jsp 的运行效果

在该页面中输入续写的内容后，单击"提交信息"按钮，save.jsp 程序获取客户续写的小说内容，并保存到 content 中。然后以 name 为文件名参数，创建 File 对象 f，把客户续写的内容 content 保存到 f 代表的文件中，并将其输出到客户端。save.jsp 程序的源代码如下：

<! -- 例程 6 - 14 save.jsp -- >

```
< % @ page contentType = "text/html;charset = GB2312"  % >
< % @ page isThreadSafe = "false"  % >
< % @ page import  = "java.io. * "  % >
```

```
<HTML>
<BODY>
  <%!
     String writeContent(File f,String s)
     {
       try{
            RandomAccessFile out = new RandomAccessFile(f,"rw");
            out.seek(out.length());      //定位到文件的末尾
            out.writeUTF(s);
            out.close();
            return "内容已成功写入到文件";
          }
       catch(IOException e)
          {
             return "不能写入到文件";
          }
     }
  %>
  <%--获取客户提交的小说的名字--%>
  <%
     String name = (String)session.getAttribute("name");
     byte c[] = name.getBytes("ISO-8859-1");
     name = new String(c);
    //获取客户续写的内容
     String content = (String)request.getParameter("messages");
     if(content == null)
       {
         content = "";
       }
   %>
  <%
     File storyFileDir = new File("E:/code/6","test");
     storyFileDir.mkdir();
     File f = new File(storyFileDir,name);
     String message = writeContent(f,content);
     byte d[] = content.getBytes("ISO-8859-1"); //输出时,要将字符串进行编码
     content = new String(d);
     out.print(content);
  %>
</BODY>
</HTML>
```

【说明】

本例主要是创建 RandomAccessFile 类型对象,然后用该对象的 seek(int length)方法定位文件指针,用 readUTF()方法读取文件,用 writeUTF(String str)方法写文件。

6.4 文件的操作

在编写网站应用程序的过程中,有许多地方都要对文件进行操作,如文件的上传下载、文件的浏览显示、创建删除目录等。本节将对 JSP 中文件操作的应用作一些介绍。

6.4.1 文件上传

文件上传是指由 JSP 获得客户端发出的输入流，再从这个输入流中读取指定的文件，然后把文件保存到指定的位置。

【例 6-9】 将客户端的文件上传。

本例由两个页面实现文件上传，upload.jsp 页面提供窗口，客户在此窗口选择要上传的文件名，将要上传的文件内容提交给 accept.jsp 页面，accept.jsp 页面获取上传文件的内容，并保存到服务器的 E:\code\6\test\to.txt 文件中。本程序源代码存放于本书配套素材中的相应文件中（文件路径为 code\6\）。

upload.jsp 程序的源代码如下：

```
<! -- 例程 6-15 upload.jsp -->

<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY>
<P>选择要上传的文件:<BR>
<FORM action="accept.jsp" method="post" ENCTYPE="multipart/form-data">
    <INPUT type=file    name="boy" size="16" ><br>
    <INPUT type="submit" name="g" value="上传">
</FORM>
</BODY>
</HTML>
```

图 6-22 upload.jsp 的运行效果

该程序运行后显示如图 6-22 所示的效果。

```
<! -- 例程 6-16 accept.jsp -->

<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import ="java.io.*" %>
<HTML>
<BODY>
  <%
    try{
        InputStream in=request.getInputStream();
        FileOutputStream ou=new FileOutputStream("E:/code/6/test/to.txt");
        byte b[]=new byte[1000];     //每次读取的字节保存在该字节数组中
        int n;                       //保存实际读取的字节数
        while((n=in.read(b))!=-1)
          {
            ou.write(b,0,n);
          }
        ou.close();
        in.close();
      }
    catch(IOException ee) { }
    out.print("文件已上传");
  %>
</BODY>
</HTML>
```

专家点拨：要上传文件，表单的 ENCTYPE 属性必须设置为 multipart/form-data，另外表单中还必须包含一个 File 类型的数据接受框，用此框接受文件名的录入。通过 request 对象的 getInputStream()方法获取客户端上传的全部数据(包括文件的内容和表单域的数据)。

6.4.2 文件下载

文件下载是指直接下载网上指定的文件。

【例 6-10】 下载服务器上的 E:/code/6/test/hlm. doc 文件。

本程序由两个页面构成：download. jsp 页面超链接到下载页面 loadFile. jsp。loadFile. jsp 页面实现文件下载功能。本程序源代码存放于本书配套素材中的相应文件中(文件路径为 code\6\)。

该程序的源代码如下：

```
<!-- 例程 6-17 download.jsp -->

<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
 <BODY>
  <P>单击超链接下载压缩文档<BR>
  <a href="loadFile.jsp">下载文件 hlm.doc </a>
 </BODY>
</HTML>
```

```
<!-- 例程 6-18 loadFile.jsp -->

<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import ="java.io.*" %>
<HTML>
<BODY>
  <%
    File fileLoad=null;
    fileLoad=new File("E:/code/6/test/hlm.doc");         //要下载的文件
    response.setHeader("Content-disposition","attachment;filename="+"hlm.doc");
  //客户使用对话框保存文件
    response.setContentType("application/msword");       //通知客户下载文件的 MIME 类型
    long fileLength=fileLoad.length();
    String length=String.valueOf(fileLength);
    response.setHeader("Content_Length",length);         //通知客户文件的长度
    FileInputStream in=new FileInputStream(fileLoad);    //读取文件并发送给客户下载
    OutputStream ou=response.getOutputStream();          //获得指向客户的输出流
    byte b[]=new byte[500];                    //每次试图从文件中读取 500 个字节到数组 b 中
    int n=0;                                             //每次从文件中读取的实际字节数
    while((n=in.read(b))!=-1)
      {
        ou.write(b,0,n);
      }
    ou.close();
    in.close();
   %>
```

```
  </BODY>
</HTML>
```

download.jsp 程序运行后显示如图 6-23 所示的效果。

在该页面中单击“下载文件 hlm.doc”链接，程序超链接到下载页面 loadFile.jsp，执行后显示如图 6-24 所示的下载对话框。

单击超链接下载压缩文档
下载文件hlm.doc

图 6-23　download.jsp 的运行效果

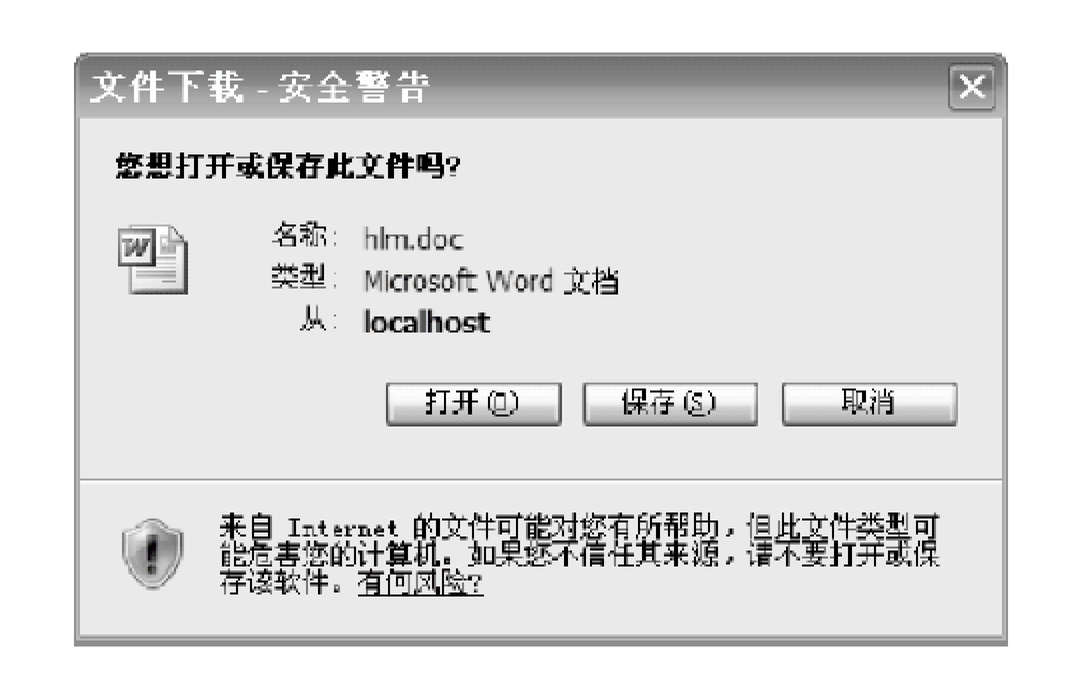


图 6-24　loadFile.jsp 运行后的下载对话框

专家点拨：要下载服务器上的文件，首先通过输入流把文件读到缓冲区中，然后把缓冲区中的数据写到 getOutputStream()方法所指向的客户端。

6.4.3　文件的分页显示

当要读取一个较大的文件时，比如，客户在网上读一部小说，就要采用分页读取文件。在 JSP 中可采用 session 对象来实现文件的分页读取。

【例 6-11】　分页读取文件。

在本例的 JSP 页面中，提供两种选择：一种是从文件头开始读（重新打开输入流）；另一种是接着文件上一次读取数据的位置开始读（使用源输入流）。本程序源代码存放于本书配套素材中的相应文件中（文件路径为 code\6\）。

该程序的源代码如下：

```
                    <! -- 例程 6 - 19 fyread.jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<%@ page import = "java.io.*" %>
<HTML>
 <BODY>
 <center>
  <%!                                         //对字符串进行回压流处理的方法
    public String getString(String content)
    {
       try{
           StringReader in = new StringReader(content);     //指向字符串的字符流
           PushbackReader push = new PushbackReader(in);    //回压流
           StringBuffer stringbuffer = new StringBuffer(); //缓冲字符串对象
           int c;
           char b[] = new char[1];
           while((c = push.read(b,0,1))! = - 1)            //读取 1 个字符放入字符数组 b
```

```
                {
                  String s = new String(b);
                  if(s.equals("<"))
                     {
                        push.unread('&');
                        push.read(b,0,1);          //push 读出被回压的字符字节,放入数组 b
                        stringbuffer.append(new String(b));
                        push.unread('L');
                        push.read(b,0,1);          //push 读出被回压的字符字节,放入数组 b
                        stringbuffer.append(new String(b));
                        push.unread('T');
                        push.read(b,0,1);          //push 读出被回压的字符字节,放入数组 b
                        stringbuffer.append(new String(b));
                      }
                   else if(s.equals(">"))
                         {
                           push.unread('&');
                           push.read(b,0,1);
                           stringbuffer.append(new String(b));
                           push.unread('G');
                           push.read(b,0,1);
                           stringbuffer.append(new String(b));
                           push.unread('T');
                           push.read(b,0,1);
                           stringbuffer.append(new String(b));
                         }
                        else if(s.equals("\n"))
                              {
                                stringbuffer.append("<BR>");
                              }
                           else
                              {
                                 stringbuffer.append(s);
                              }
                   }
                 push.close();
                 in.close();
                 return new String(stringbuffer);    //返回处理后的字符串
               }
            catch(IOException e)
               {
                 return content = new String("不能读取内容");
               }
          }
%>
<FORM action = "" method = "post" name = form>
   <Input type = submit name = "g" value = "从文件头开始读">
</FORM>
<FORM action = "" method = "post" name = form>
   <Input type = submit name = "g" value = "续读取文件的下 10 行">
</FORM>
<%
String s = request.getParameter("g");
    if(s == null)
```

```
        {s = "";}
      byte b[ ] = s.getBytes("ISO-8859-1");
      s = new String(b);
      File f = null;
      FileReader in = null;
      BufferedReader buffer = null;
      if(s.equals("从文件头开始读"))
        {
          f = new File("E:/code/6/test/tscs.txt");     //重新打开输入流
          in = new FileReader(f);        //重建文件输入流,流指向文件的开头
          buffer = new BufferedReader(in);    //重建缓冲流,流指向文件的开头
          session.setAttribute("file",f);
          session.setAttribute("in",in);
          session.setAttribute("buffer",buffer); //将输入流保存到客户的 session 对象中
        }
      else
        {
          try
           {
           String str = null; int i = 1;
           f = (File)session.getAttribute("file");
           in = (FileReader)session.getAttribute("in");
           buffer = (BufferedReader)session.getAttribute("buffer");
           while(((i<=10)&&(str = buffer.readLine())!= null)) //每次从流中读 10 行数据
                {
                  str = getString(str);
                  out.print("<Font size = 2><BR>" + str + "</Font>");
                  i++ ;
                }
            }
         catch(Exception e)
            {
              out.print("读取出现问题,请单击重新读取按钮" + e);
            }
       }
   %>
 </center>
 </BODY>
</HTML>
```

该程序运行后的效果如图 6-25 所示。

图 6-25　fyread.jsp 的运行效果

在该程序中,每次重读文件时,将重建的输入流 buffer 保存到 session 对象中,当希望接上次续读文件后 10 行时,从 session 对象中取出源输入流 buffer,然后在上次的流指针处读取文件。

6.4.4 创建和删除目录

文件是用来保存数据的,目录是管理文件的特殊机制,同类文件保存在同一个目录下可以简化文件管理。因此,掌握文件和目录的操作对于编程人员是必需的。

【例 6-12】 目录的创建与删除。

本例的主要功能是,如果没有目录文件就创建一个新的目录文件,如果目录文件存在就删除该目录文件,该程序的源代码如下。本程序源代码存放于本书配套素材中的相应文件中(文件路径为 code\6\)。

```
<! -- 例程 6 - 20 mulu.jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<%@ page import = "java.io.*" %>
<html>
 <body>
  <%
     File d = new File("E:/code/6","mulu");            //设置要建立的目录名及路径
     if (d.exists())                                    //检查目录是否存在
     {
       d.delete();                                      //删除目录
       out.println("mulu 目录已存在,已被删除");
     }
      else
      {
       d.mkdir();                                       //建立目录
       out.println("mulu 目录不存在,已建立");
      }
  %>
</body>
</html>
```

在本例中,File 对象调用 mkdir()方法创建一个目录,如果创建成功就返回 true,否则就返回 flase。File 对象调用 delete()方法删除一个目录,必须注意的是,如果 File 对象表示一个目录,那么该目录必须是一个空目录才能被删除。

6.5 上机指导与练习

6.5.1 列出 C 盘根目录下的所有子目录和文件

1. 练习目标

(1) 练习创建一个对象,并获得该对象包含的所有目录和文件。

(2) 掌握使用条件语句将文件和目录分开。

2. 练习指导

(1) 以 C:/为参数,创建对象 dir。

(2) 调用 listFile()方法,获取对象 dir 所包含的对象数组(文件和目录构成的对象):file[]。

(3) 输出所包含的子目录。

(4) 输出所包含的文件。

该程序的源代码如下。

```
<! -- 例程 6-21 listfile.jsp -->
<%@ page contentType = "text/html;charset = GB2312" %>
<%@ page import = "java.io.*" %>
<HTML>
<BODY><FONT Size = 2>
   <%
      File dir = new File("C:/");
      File file[] = dir.listFiles();
    %>
  <BR>目录列表:
  <%
     for(int i = 0;i<file.length;i++)
       {
         if (file[i].isDirectory())
            out.print("<BR>" + file[i].toString());
       }
   %>
  <P>文件列表:
  <%
     for(int i = 0;i<file.length;i++)
       {
          if(file[i].isFile())
             out.print("<BR>" + file[i].toString());
       }
   %>
</FONT>
</BODY>
</HTML>
```

知识点:如果 File 对象是一个目录,调用 listFile()方法时,返回该目录下的全部文件和子目录。isFile()方法判断 File 对象是否是文件,isDirectory()方法判断当前 File 对象是否是目录。本程序源代码存放于本书配套素材中的相应文件中(文件路径为 code\6\上机指导)。

6.5.2 列出 E:/code/6 目录下所有的 JSP 文件

1. 练习目标

(1) 熟悉类 FileJSP 的定义。

(2) 掌握对目录下的文件进行筛选的方法。

2. 练习指导

(1) 以 E:/code/6 为参数,构造目录对象 dir。
(2) 以 jsp 为文件后缀,构造文件筛选条件 file_jsp。
(3) 以筛选条件 file_jsp 对目录对象 dir 进行筛选,得到文件数组 file_name。
(4) 输出文件数组。

该程序的源代码如下。

```
<!-- 例程 6-22 jsptype.jsp -->

<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.io.*" %>
 <%!
   class FileJSP implements FilenameFilter
   {
     String str=null;
     FileJSP(String str)
     {
       this.str="."+str;
     }
     public boolean accept(File dir,String name)
     {
       return name.endsWith(str);
     }
   }
 %>
  <P>"E:/code/6"目录下,所有的 jsp 文件文件:
  <%
     File dir=new File("E:/code/6");
     FileJSP file_jsp=new FileJSP("jsp");
     String file_name[]=dir.list(file_jsp);
     for(int i=0;i<file_name.length;i++)
       {
         out.print("<BR>"+file_name[i]);
       }
  %>
```

知识点:实现接口 FilenameFilter 的 accept(File dir,String name)方法,该方法用文件后缀为 str 为过滤器,对 dir 目录下的文件 name 进行筛选。本程序源代码存放于本书配套素材中的相应文件中(文件路径为 code\6\上机指导)。

6.5.3 将客户端的文件上传到服务器

1. 练习目标

(1) 掌握文件的上传。
(2) 掌握图片文件的显示。

2. 练习指导

本程序将客户端的文件上传到服务器(去掉表单域信息),若是图像文件,则查看该图像。在该程序中将客户上传的全部信息保存到一个临时文件中,用客户的 session 的 ID 作为临时文件的名字。程序由 3 个页面构成。image. jsp 页面将上传的文件提交给 acceptFile. jsp 页面。acceptFile. jsp 页面获取上传文件的内容,将其保存到服务器的 E:/code/6/test 目录中。showImage. jsp 页面查看上传的图片文件。

1) 程序 image. jsp 的功能及代码

创建一个表单,该表单包含一个 File 类型的数据框。使用该数据框录入上传的文件名。

```
<! -- 例程 6 - 23 image. jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<HTML>
 <BODY>
   <P>选择要上传的文件:<BR>
     <FORM action = "acceptFile. jsp" method = "post" ENCTYPE = "multipart/form - data">
       <INPUT type = file name = "boy" size = "16" ><br>
       <INPUT type = "submit" name = "g" value = "上传">
     </FORM>
 </BODY>
</HTML>
```

2) 程序 acceptfile. jsp 的功能及代码

(1) 获取客户上传的全部信息,并保存到输入流 in 中。

(2) 以临时文件 f1 为参数,创建一个输出流 ou。

(3) 读取输入流(in)的全部信息,写入到输出流(ou)中,即写入 f1 文件中。

(4) 从文件 f1 中的第 2 行中,获取上传文件的文件名,并保存到变量 fileName 中。

(5) 把文件名 fileName 保存到 session 对象中(fileName 用于 showImage. jsp 页面)。

(6) 以 fileName 为参数,创建随机文件 f2。

(7) 找出文件 f1 的第 4 行结尾处的字节偏移值,其值保存在 forthEndPosition 中。

(8) 找出文件 f1 的倒数第 6 行结尾处的字节偏移值,其值保存在 endPosition 中。

(9) 将 f1 的文件指针指向 forthEndPosition 处。

(10) 从 f1 文件中,读取起始字节为 forthEndPosition 到结束字节为 endPosition 的数据,把读取的全部数据保存到 f2 文件中。

(11) 删除临时文件 f1。

(12) 创建一个表单,该表单将数据提交给 showImage. jsp 页面。

```
<! -- 例程 6 - 24 acceptfile. jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<%@ page import = "java. io. * " %>
<HTML>
<BODY>
  <%
     String fileName = null;
```

```
try
{
 InputStream in = request.getInputStream(); //获取客户上传的全部信息 in
 String tempFileName = (String)session.getId();
   //用客户的 session 对象的 ID 建立一个临时文件
 File f1 = new File("C:/",tempFileName);
  //建立临时文件 f1,用 f1 保存客户上传的全部信息
 FileOutputStream ou = new FileOutputStream(f1);
 byte b[ ] = new byte[1000];
 int n;
 while( (n = in.read(b))! = - 1)                    //将输入流 in 写入输出流 ou 中
      {
         ou.write(b,0,n);
      }
    ou.close();
    in.close();
    //从 f1 中获取上传文件的名字和上传文件的内容,去掉表单域信息
    RandomAccessFile random = new RandomAccessFile(f1,"r");
   //指针定位在文件开头读出 f1 的第 2 行字符串,保存到 secondLine 中
    int second = 1;
    String secondLine = null;
    while(second <= 2)
       {
         secondLine = random.readLine();
         second++;
       }
    //获取第 2 行中目录符号'\'最后出现的位置
    int position = secondLine.lastIndexOf('\\');
    //获取客户上传的文件的名字,不包含文件路径
    fileName = secondLine.substring(position + 1,secondLine.length() - 1);
    byte cc[ ] = fileName.getBytes("ISO - 8859 - 1");
    fileName = new String(cc);
    session.setAttribute("name",fileName);
    File f2 = new File("E:/code/6/test",fileName);
    RandomAccessFile random2 = new RandomAccessFile(f2,"rw");
    random.seek(0);                                  //再定位到文件 f1 的开头
    long forthEndPosition = 0 ;
    int forth = 1;                                   //记录读取过的行数
    while((n = random.readByte())! = - 1&&(forth <= 4))
      {
        if(n == '\n')
          {
            forthEndPosition = random.getFilePointer();
            forth++;
          }
      }
    random.seek(random.length());                    //游标移到文件最后一个字节处
    long endPosition = random.getFilePointer();
    long mark = endPosition;
    int j = 1;                                       //记录读取过的行数
    while((mark >= 0)&&(j <= 6))
```

```
            {
              mark -- ;
              random.seek(mark);                         //指针后退一个字节
              n = random.readByte();                     //读取当前指针指向的一个字节
              if(n == '\n')
                {
                  endPosition = random.getFilePointer(); //获取当前游标指针值
                  j++ ;
                }
            }
          random.seek(forthEndPosition);
          long startPoint = random.getFilePointer();
          while(startPoint < endPosition - 1)
            {
              n = random.readByte();
              random2.write(n);
              startPoint = random.getFilePointer();
            }
          random2.close();
          random.close();
          f1.delete();                                   //删除临时文件
        }
      catch(IOException ee) { }
      out.print("上传文件保存在服务器的 E:/code/6/test" + fileName);
  %>
  <p>
  <FORM action = "showImage.jsp" >
      查看上传的图像效果:
      <Input type = submit value = "查看">
  </FORM>
</BODY>
</HTML>
```

3）程序 showimage.jsp 的功能及代码

（1）获取 session 对象中的文件名 fileName。

（2）将该文件的图像输出到客户端。

<! -- 例程 6 - 25 showimage.jsp -- >

```
<%@ page contentType = "text/html;charset = GB2312"  %>
<HTML>
<BODY>
  <%
      String pic = (String)session.getAttribute("name");
      out.print(pic);
      out.print("< image src = " + pic + ">");
  %>
</BODY>
</HTML>
```

本程序源代码存放于本书配套素材中的相应文件中(文件路径为 code\6\上机指导)。

本章小结

本章主要介绍了在页面中对文件的操作方式,文件操作是网站编程的重要内容之一。在编写网站应用程序的过程中,有许多地方要对文件进行操作。通过本章的学习,应该掌握数据流的应用及文件操作的应用。

习题 6

一、简答题

1. 用来生成 File 对象的构造函数有哪些?

2. 按照数据流成分划分,可将数据流分为哪几类?

3. 简述对文件进行随机读写的类。

二、操作编程题

1. 编写 JSP 页面程序,完成读写文本文件的功能。

2. 编写 JSP 页面程序,列出当前目录中的所有文件和子目录,并对文件和子目录进行统计。

第7章 JSP与JavaBean

JSP作为一个很好的动态网站开发语言得到了越来越广泛的应用，在各类JSP应用程序中，JSP+ JavaBean的组合成为一种最常见的JSP程序的标准。本章将介绍JavaBean的编写和使用方法。

本章主要内容：

- JavaBean组件的开发；
- 在JSP中使用JavaBean；
- JavaBean的范围；
- 通过JavaBean访问数据库。

7.1 JavaBean介绍

JavaBean是一段特殊的Java类，其最初的目的是将可以重复使用的软件代码打包，特别是用于帮助厂家开发在综合开发环境(IDE)下使用的Java软件部件，如今，JavaBean部件框架已经扩展为企业版的Bean(EJB)。

7.1.1 什么是JavaBean

JSP和JavaBean结合使用是目前比较流行的Web开发技术，JavaBean是一个可以重复使用的、跨平台的组件，描述了Java的软件组件模型，有点类似于Microsoft的COM组件概念。在Java模型中，通过JavaBean可以无限扩充Java程序的功能，快速生成新的应用程序。

JavaBean一般分为可视化组件和非可视化组件两种。可视化组件可以是简单的GUI元素，如按钮或文本框，也可以是复杂的，如报表组件；非可视化组件没有GUI表现形式，用于封装业务逻辑、数据库操作等，与JSP配合使用，很好地实现了业务逻辑与显示层的分离，使系统具有更好的健壮性和灵活性。现在，JavaBean被更多地应用在非可视领域。

通常一个标准的JavaBean具有以下特性。

(1) 易于维护、使用和编写。

(2) 可实现代码的重用性。

(3) 可移植性强，但仅限于Java工作平台。

(4) 便于传输，不限于本地还是网络。

(5) 可以以其他部件的模式进行工作。

专家点拨：JavaBean组件与企业级JavaBean(Enterprise JavaBean,EJB)组件完全不同。EJB是J2EE的核心，是一个用来创建分布式应用、服务器端以及基于Java应用的功能强大的组件模型。JavaBean组件主要用于存储状态信息，而EJB组件可以存储业务逻辑。

7.1.2 JavaBean的组成

一个JavaBean由3部分组成，下面分别进行介绍。

1. 属性(Properties)

JavaBean提供了高层次的属性概念，属性在JavaBean中不只是传统的面向对象的概念里的属性，它同时还得到了属性读取和属性写入的API支持。属性值可以通过调用适当的Bean()方法进行。例如，Bean有一个名字属性，这个属性的值可能需要调用String getName()方法读取，而写入属性值可能需要调用void setName(String str)方法。

每个JavaBean属性都应该遵循方法的命名规则，这样应用程序构造器工具和最终用户才能找到JavaBean提供的属性，然后查询或修改属性值，并对Bean进行操作。JavaBean还可以对属性值的改变做出及时的反应，如一个显示当前时间的JavaBean，如果改变时钟的时区属性，则时钟会立即重画，显示当前指定时区的时间。

2. 方法(Method)

JavaBean中的方法就是通常的Java方法，它可以从其他组件或在脚本环境中调用。默认情况下，所有Bean的公有方法都可以被外部调用，但Bean一般只会引出其公有方法的一个子集。

由于JavaBean本身是Java对象，调用这个对象的方法是与其交互作用的唯一途径。JavaBean严格遵守面向对象的类设计逻辑，不让外部世界访问其任何字段(没有Public字段)。这样，方法调用是接触Bean的唯一途径。

3. 事件(Event)

Bean与其他组件交流信息的主要方式是发送和接收事件。可以将Bean的事件支持功能看做是集成电路中的输入输出引脚：工程师将引脚连接在一起组成系统，让组件进行通信。有些引脚用于输入，有些引脚用于输出，相当于事件模型中的发送事件和接收事件。

事件为JavaBean组件提供了一种发送通知给其他组件的方法。在AWT事件模型中，一个事件源可以注册事件监听器对象。当事件源检测到发生了某种事件时，它将调用事件监听器对象中的一个适当的事件处理方法来处理这个事件。

7.2 编写JavaBean

虽然JavaBean和Java之间已经有了明确的界限，但是在某些方面JavaBean和Java之间仍然存在很容易混淆的地方，如重用，Java语言也可以为用户创建可重用的对象，但它没

有管理这些对象相互作用的规则或标准，用户可以使用在 Java 中预先建立好的对象，但必须具有对象在代码层次上的接口的丰富知识。而对于 JavaBean，用户可以在应用程序构造器工具中使用各种 JavaBean 组件，而不需要编写任何代码。这种同时使用多个组件而不考虑其初始化情况的功能是对当前 Java 模型的重要扩展，所以也可以说，JavaBean 是在组件技术上对 Java 语言的扩展。

7.2.1 开发 JavaBean 组件

JavaBean 是一种组件技术，可以将内部的动作（如事务逻辑、数据库操作等）封装起来，用户看不到它是如何运行的，它只提供最小限度的属性接口供 JSP 程序使用，实现了业务逻辑和前台程序的分离。操作过程往往是先开发需要的 JavaBean，再在适当的时候进行调用。

JavaBean 作为一个特殊的类，具有自己独有的特性，为了能使用 Bean 的应用程序构建工具（比如 JSP 引擎）知道 Bean 的属性和方法，只需在类的方法命名上遵守以下规则：

- JavaBean 类必须有一个没有参数的构造函数。
- JavaBean 类所有的属性最好定义为私有的（Private）。
- JavaBean 类中方法的访问属性都必须是 Public 的。
- JavaBean 类中定义函数 setXxx() 和 getXxx()来对属性进行操作。其中，Xxx 是首字母大写的私有变量名称。
- 对于 boolean 类型的成员变量，即布尔逻辑类型的属性，允许使用 is 代替上面的 get 和 set。

【例 7-1】 创建一个简单的 JavaBean 程序。

本例用一个实例程序 FirstJavaBean. java 来介绍如何开发 JavaBean 组件，其源代码如下。本例源代码存放于本书配套素材中的 FirstJavaBean. java 文件中（文件路径为 code\7）。

<! -- 例程 7 - 1 FirstJavaBean. java -->

```
package bean;

public class FirstJavaBean {
    private String FirstProperty = new String("");
    public FirstJavaBean() {
    }
    public String getFirstProperty() {
        return FirstProperty;
    }
    public void setFirstProperty(String value) {
        FirstProperty = value;
    }
    public static void main(String[ ] args)
    {
        System.out.println("第一个 JavaBean!");
    }
}
```

该程序运行的效果如图 7-1 所示。

<terminated> FirstJavaBean [Java Application]

第一个JavaBean!

图 7-1 程序运行效果

本程序是一个很典型的 JavaBean 代表，FirstProperty 是其中的一个属性(Property)，外部通过 get()/set()方法可以对这个属性进行操作。main()方法是为了测试程序用的，在编写 JavaBean 时可以先不必加入到 JSP 程序中调用，而直接用 main()方法来进行调试，调试好以后就可以在 JSP 程序中调用了。

7.2.2　在页面中使用 JavaBean 组件

JSP 自身没有实现 HTML 代码与 Java 代码的完全分离，网页设计人员和 Java 编程人员操作同一个 JSP 文件，不易维护和管理。在 JSP 页面中是通过操作指令：<jsp:useBean>、<jsp:setProperty>和<jsp:getProperty>来应用 JavaBean 的，它们分别用于创建和查找 JavaBean 的实例对象、设置 JavaBean 对象的属性及读取 JavaBean 对象的属性。

1. <jsp:useBean>指令

<jsp:useBean>指令用来定义生成和使用 Bean 的环境，即如果使用<jsp:useBean>，就可以定义 Bean 的名称、类型以及使用期限等内容。

在 JSP 页面中，使用 JavaBean 前首先要声明 JavaBean。JavaBean 的声明通过动作来实现，其语法格式如下：

```
<jsp: useBean id="名称" scope="范围" class="类名称" type="类类型" beanName="Bean的名称">
```

【说明】

(1) id：用于创建 JavaBean 实例的名字，在所定义的范围中确认 Bean 的变量，能在后面的程序中使用此变量名来分辨不同的 Bean。如果 Bean 已经在别的<jsp:useBean>中创建，那么这个 id 的值必须与原来的那个 id 值一致。

(2) scope：指定了 JavaBean 的作用范围(也叫生存期或生命周期)，可以有如下几个属性值。

- application：在整个应用程序范围内有效。
- page：在当前页面范围内有效。
- request：在当前请求范围内有效。
- session：在当前会话范围内有效。

(3) class：用来指定 JSP 引擎查找 JavaBean 代码的路径，一般是 JavaBean 的类名。

(4) type：定义 Bean 对象的类型。

(5) beanName：使用和 class 参数相似的概念来定义 Bean。如果不使用 class 参数，就要定义 beanName 参数，这是为了完成 Bean 的内部代码而使用的属性。

2. <jsp:setProperty>指令

<jsp:setProperty>是设定通过<jsp:useBean>定义的 Bean 对象属性的标记。<jsp:setProperty>的语法格式有以下 4 种。

(1) 字符串常量：

```
<jsp:setProperty name="beanName" property="propName" value="string constant"/>
```

（2）请求参数：

```
<jsp:setProperty name = "beanName" property = "propName" param = "paramName"/>
```

（3）匹配 Bean 中指定的属性：

```
<jsp:setProperty name = "beanName" property = "propName"/>
```

（4）表达式：

```
<jsp:setProperty name = "beanName" property = "propName" value = "<% = expression %>"/>
```

【说明】

（1）name：指定要设置的 JavaBean 名称。这个 JavaBean 必须首先使用<jsp:useBean>操作指令来实例化。

（2）property：用于指定 JavaBean 需要定制的属性名称，如果属性值为"*"，则会使用提交表单中的所有值来填充 JavaBean 属性。

（3）param：使用 Request 中的一个参数值来指定 Bean 中的一个属性值。

专家点拨： 如果 Bean 属性和 Request 参数的名字不同，则必须指定 property 和 param，如果它们同名，那么只需指明 property 就可以了。

（4）value：使用指定的值来设定 Bean 属性，这个值可以是字符串，也可以是表达式。

3. <jsp:getProperty>指令

<jsp:getProperty>用于从一个 JavaBean 中获取某个属性的值，无论原来这个属性是什么类型，都将被转换为一个 String 类型的值。其语法格式如下：

```
<jsp:getProperty name = "beanName" property = "propName"/>
```

【说明】

（1）name：指定 JavaBean 的名称，并且 JavaBean 组件对象必须已经使用<jsp:useBean>操作指令实例化了。

（2）property：用于指定要读取的 JavaBean 组件对象的属性名称。

【例 7-2】 建立一个 JSP+JavaBean 程序。

本程序是一个注册信息提交的程序，通过注册界面输入相关信息后，将注册的信息显示出来。该程序的相关代码如下。本例源代码存放于本书配套素材中的相应文件中（文件路径为 code\7）。

（1）创建一个 JavaBean 组件 StudentBean.java。

```
<! -- 例程 7 - 2 StudentBean.java -->
```

```
package bean;

public class StudentBean {
    private String name;
    private boolean sex;
    private int age;
    public String getName() {
        return name;
```

```
    }
    public void setName(String name) {
        this.name = name;
    }
    public boolean isSex() {
        return sex;
    }
    public void setSex(boolean sex) {
        this.sex = sex;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

（2）创建一个 studentreg.html 文件。

```
<!-- 例程 7-3 studentreg.html -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>studentreg.html</title>
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="this is my page">
    <meta http-equiv="content-type" content="text/html; charset=gb2312">
  </head>
  <body>
  <br>
  <form method="post" action="studentreg.jsp" name="form1">
  <table width="400" align="center" cellpadding="0" cellspacing="0" border="1" bgColor=
"#cad728">
  <tr>
    <td colspan="2" align="center">学生注册</td>
  </tr>
  <tr>
    <td width="30%">姓名：</td>
    <td width="70%"><input type="text" name="name"/></td>
  </tr>
  <tr>
    <td>性别：</td>
    <td>
        <input type="radio" value="1" checked="checked" name="sex"/>男
        <input type="radio" value="0" name="sex" />女
    </td>
  </tr>
  <tr>
    <td>年龄：</td>
    <td><input type="text" name="age"/></td>
  </tr>
```

```
  <tr>
    <td colspan="2" align="center">
        <input type="submit" value="注册"/>  
        <input type="reset" value="重置"/>
    </td>
  </tr>
  </table>
  </form>
  </body>
</html>
```

该文件运行后显示效果如图 7-2 所示。



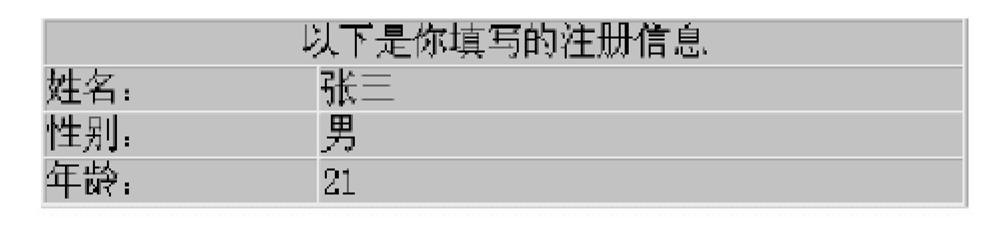
图 7-2 studentreg.html 的运行效果

(3) 创建一个 studentreg.jsp,使用 StudentBean 组件。

```
<!-- 例程 7-4 studentreg.jsp -->
<%@ page language="java" pageEncoding="gb2312"%>
<%
      request.setCharacterEncoding("gb2312");
    <%--该条语句的作用是将 request 的编码字符集设为 gb2312,如果没有这条语句,页面上将
无法正常显示汉字。--%>
%>
<jsp:useBean id="stuBean" scope="page" class="bean.StudentBean"/>
<jsp:setProperty name="stuBean" property="*" />
    <%--这两句通过<jsp:useBean>指令初始化了一个名为 stuBean 的 stuBean 实例,该指令中的
scope 属性定义了 JavaBean 的范围为 page,其后的<jsp:setProperty>指令将用户请求中的参数值赋
给 logBean 中相应的属性,使用特殊的属性取值"*",可以一次性设置 JavaBean 中所有的属性。其后
的<jsp:getProperty>指令取出 stuBean 中保存的数据。--%>
<%
String path = request.getContextPath();
String basePath = request.getScheme() + "://" + request.getServerName() + ":" + request.
getServerPort() + path + "/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html:html language="true">
  <head>
    <html:base />
    <title>注册成功</title>
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
  </head>
  <body>
```

```
  <table width = "400" align = "center" cellpadding = "0" cellspacing = "0" border = "1" bgColor =
"#cad728">
  <tr>
    <td colspan = "2" align = "center">以下是你填写的注册信息</td>
  </tr>
  <tr>
    <td width = "30%">姓名:</td>
    <td width = "70%"><jsp:getProperty name = "stuBean" property = "name"/></td>
    //通过<jsp:getProperty>指令获得 stuBean 实例的 name 属性值
  </tr>
  <tr>
    <td>性别:</td>
    <td>
<% if (stuBean.isSex())
      out.print("女");
    else
      out.print("男");
 %>
    <%-- 该段代码是通过 stuBean.isSex()获得 stuBean 实例的 sex 属性值,如果 sex 为 1 输出
"女",否则输出"男" --%>
    </td>
  </tr>
  <tr>
    <td>年龄:</td>
    <td><jsp:getProperty name = "stuBean" property = "age"/></td>
  </tr>
</table>
  </body>
    </html:html>
```

该程序运行后显示效果如图 7-3 所示。



以下是你填写的注册信息	
姓名:	张三
性别:	男
年龄:	21

图 7-3　studentreg.jsp 的运行效果

【说明】

(1) JavaBean 应放置在 JSP 页面的类装载器或其父级类装载器所能装载的目录中,通常放置于 Web 应用程序下的 WEB-INF\classes 目录中。

(2) 有些版本的 Tomcat 不会自动重新加载修改过的 JavaBean,如果 JSP 页面加载 JavaBean 后又修改和重新编译了 JavaBean 程序,那么需要修改 JSP 页面或者重新启动 Tomcat。

(3) JavaBean 必须带有包名,不能用默认包名。

(4) 在选择存储 JavaBean 的域范围时,如果使用 request 域能够满足需求,则不要使用 session 域。

7.3　JavaBean 的范围

在 7.2 节介绍了 JavaBean 的 Scope 属性,该属性具有 4 个可能的值,分别为 page、request、session 和 application,使得 JavaBean 组件对于不同的任务具有不同的生命周期和

不同的使用范围。

7.3.1 page 范围

page 范围的生命周期和作用范围在 4 种类型的 JavaBean 组件中是最小的。当 JavaBean 的 Scope 属性被设为 page 时，表示该 JavaBean 的生命周期只在一个页面内，即为 JSP 程序的运行周期。当 JSP 程序运行结束，该 JavaBean 组件的生命周期也就结束了。它无法在别的 JSP 程序中起作用，对应于不同的客户端请求服务器都会创建新的 JavaBean 组件对象，而且一旦客户端的请求执行完毕，该 JavaBean 对象会马上注销，无法供其他客户端请求使用。

【例 7-3】 页面计数器。

本程序主要用于统计页面被访问的次数，相关程序的源代码如下。本例源代码存放于本书配套素材中的相应文件中(文件路径为 code\7)。

(1) 计数器程序 Counter.java。

```
<!-- 例程 7-5 Counter.java -->

package bean;
public class Counter {
    private int count;
    public Counter() {
        count = 0;
    }
    public int getCount() {
        count ++ ;
        return count;
    }
    public void setCount(int count) {
        this.count = count;
    }
}
```

(2) 计数器显示页面 count_page.jsp 文件。

```
<! -- 例程 7-6 count_page.jsp -->

<%@ page language="java" import="java.util.*" pageEncoding="GB2312"%>
<%
String path = request.getContextPath();
String basePath = request.getScheme() + "://" + request.getServerName() + ":" + request.
getServerPort() + path + "/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">
    <title>page 范围</title>
```

```
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
  </head>
  <body>
    <jsp:useBean id="counter" scope="page" class="bean.Counter" />
     <br>你好!你是第<b><% out.println(counter.getCount()); %></b>位访客
  </body>
</html>
```

运行后效果如图 7-4 所示。

在该程序运行过程中，刷新页面时，无论刷新多少次，显示的都是“第 1 位访客”。这是因为当刷新页面时，JSP 容器都会将以前的 JavaBean 清除，然后重新产生一个 JavaBean。因此，使用 getCount()方法取值时，总是取出值为 1。

你好！你是第**1** 位访客

图 7-4　page 范围程序运行效果

7.3.2　request 范围

request 范围的生命周期和作用范围与 JSP 的 Request 对象一样，当 JavaBean 的 Scope 属性值被设为 request 时，表示 JavaBean 在整个请求的范围内都有效，而不仅仅在一个页面内有效。

当一个 JSP 程序使用<jsp:forward>操作指令定向到另外一个 JSP 页面或使用<jsp:include>操作指令导入另外的 JSP 页面时，第一个 JSP 页面会把 Request 对象传送到下一个 JSP 页面，由于 request 范围的 JavaBean 存在于 Request 对象中，因此，JavaBean 对象也将随着 Request 对象送出，而被第二个 JSP 程序接收。这种类型的 JavaBean 对象使得 JSP 程序之间传递信息更为容易。

【例 7-4】　request 范围测试。

本例修改例 7-3 的例子，将 count_page.jsp 改名为 count_request.jsp，再增加一个新的页面 request1.jsp，查看 request 范围与 page 范围的 JavaBean 有何不同。相关程序的源代码如下。本例源代码存放于本书配套素材中的相应文件中（文件路径为 code\7）。

```
<!-- 例程 7-7 count_request.jsp -->

<%@ page language="java" import="java.util.*" pageEncoding="GB2312"%>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">
```

```
      <title>request 范围</title>
      <meta http-equiv="pragma" content="no-cache">
      <meta http-equiv="cache-control" content="no-cache">
      <meta http-equiv="expires" content="0">
      <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
      <meta http-equiv="description" content="This is my page">

    </head>
    <body>
      <jsp:useBean id="counter" scope="request" class="bean.Counter" />
       <br>你好!你是第<b><% out.println(counter.getCount()); %></b>位访客
       <p>欢迎再次访问:</p>
        <jsp:include page="request1.jsp" flush="true"/>
    </body>
</html>
```

```
<!-- 例程 7-8 request1.jsp -->

<%@ page contentType="text/html;charset=gb2312" %>
  <jsp:useBean id="req" class="bean.Request" scope="request" />
     你好!感谢你第<jsp:getProperty name="req" property="msg" />次的光临
```

程序运行后效果如图 7-5 所示。

从运行的结果可以看到，request1.jsp 已经加入到 count_request.jsp 中，而 request1.jsp 中显示的次数为 2 次，这是因为 request1.jsp 与 count_request.jsp 调用的是同一个 JavaBean，所以 JavaBean 中的 count 先是自动加 1 显示出来，然后又加 1 显示出来，因此其显示结果各为 1 和 2。

你好！你是第 1 位访客
欢迎再次访问：
你好！感谢你第 2 次的光临

图 7-5　request 范围程序运行效果

7.3.3　session 范围

session 范围的生命周期就是某个会话过程所经历的时间。当 JavaBean 的 Scope 属性值为 session 时，表示 JavaBean 可以在当前 HTTP 会话的生命周期内被所有页面访问，该 JavaBean 存在于 session 对象中。

实际上，会话过程是对于单个用户而言的，会话过程的开始以用户开始访问某个网站为标志，会话过程的结束以用户结束对该网站的访问为标志。不同的用户对应着不同的会话过程，不同的会话过程之间是互不干涉互不影响的。

假设某一个用户第一次登录某个网站的某个 JSP 页面，而这个 JSP 页面用到了一个 Scope 属性为 session 的 JavaBean，服务器会自动创建这个 JavaBean 的实例对象，并且当此用户继续访问同一个网站的其他 JSP 页面，其他的 JSP 程序又用到同一个 JavaBean 对象时，服务器不会创建新的 JavaBean 对象，而是使用已经存在的 JavaBean 对象实例。

【例 7-5】　session 范围测试。

本例再把例 7-3 的程序 count_page.jsp 作一些简单修改，只把 JavaBean 的范围改为 session，并重新命名为 count_session.jsp，观察执行结果。程序源代码如下。本例源代码存放于本书配套素材中的相应文件中(文件路径为 code\7)。

记录登录用户信息的 LoginUser.java 代码。

```
<!-- 例程 7-9 count_session.jsp -->

<%@ page language = "java" import = "java.util.*" pageEncoding = "GB2312" %>
<%
String path = request.getContextPath();
String basePath = request.getScheme() + "://" + request.getServerName() + ":" + request.
getServerPort() + path + "/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
   <head>
    <base href = "<% = basePath %>">
    <title>request 范围</title>
    <meta http-equiv = "pragma" content = "no-cache">
    <meta http-equiv = "cache-control" content = "no-cache">
    <meta http-equiv = "expires" content = "0">
    <meta http-equiv = "keywords" content = "keyword1,keyword2,keyword3">
    <meta http-equiv = "description" content = "This is my page">

  </head>
  <body>
   <jsp:useBean id = "counter" scope = "session" class = "bean.Counter" />
    <br>你好!你是第<b><% out.println(counter.getCount()); %></b>位访客
    <p>欢迎再次访问: </p>
  </body>
</html>
```

程序运行后的效果如图 7-6 所示。

你好！你是第 5 位访客
欢迎再次访问：

图 7-6　session 范围程序运行效果

第一次执行 count_session.jsp 时可见结果和例 7-4 的 count_page.jsp 一样,但当执行页面刷新时,可以看到页面上显示的次数会递增,这是对于同一个 session 的情况。若另外启动一个浏览器,执行 count_session.jsp 时,会发现页面上的数字又从 1 开始。这是因为新启动的浏览器又发起了一个新的 HTTP 会话,JSP 会创建一个新的 session 以及新的 JavaBean。

7.3.4　application 范围

当 JavaBean 的 Scope 属性被指定为 application 时,它的生命周期和 JSP 的 Application 对象具有相同的作用范围,也和 Application 对象一样使用。这个 JavaBean 的生命周期是最长的,从创建了这个 JavaBean 开始,就可以在任何时候使用相同 application 的 JSP 文件中使用这个 JavaBean。这种类型的 JavaBean 可以在多个用户之间共享全局信息。

JavaBean 在 Application 范围内,如果某个 JSP 程序使用<jsp:useBean>操作指令创建了一个 JavaBean 对象,而且这个 JavaBean 具有 Application Scope,那么这个 JavaBean 就一直在服务器的内存空间中,随时处理客户端的请求,直到服务器关闭为止,它所保存的信

息才消失，占用的系统资源才会被释放。在此期间如果有其他用户请求的 JSP 程序需要用到这个 JavaBean，服务器在执行<jsp:useBean>操作指令时并不会创建新的 JavaBean，而是创建源对象的一个同步副本，并且在副本对象上发生的改变都会影响到源对象，源对象也会做出同步的改变，不过这个改变不会影响其他已经存在的副本。

【例 7-6】 application 范围测试。

本例还是通过页面计数器的例子来说明 application 范围的 JavaBean 和其他范围有何不同。程序代码如下。本例源代码存放于本书配套素材中的相应文件中（文件路径为 code\7）。

```
<! -- 例程 7 - 10 count_session.jsp -->

<%@ page language="java" import="java.util.*" pageEncoding="GB2312"%>
<%
String path = request.getContextPath();
String basePath = request.getScheme() + "://" + request.getServerName() + ":" + request.
getServerPort() + path + "/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
   <head>
    <base href="<%=basePath%>">
    <title>application 范围</title>
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
    <!--
    <link rel="stylesheet" type="text/css" href="styles.css">
    -->
   </head>
   <body>
    <jsp:useBean id="counter" scope="application" class="bean.Counter" />
      <br>你好!你是第<b><% out.println(counter.getCount()); %></b>位访客
  </body>
</html>
```

程序显示页面运行效果如图 7-7 所示。

你好！你是第6 位访客

图 7-7 application 范围程序运行效果

第一次执行 count_application.jsp 时可见结果和 count_session.jsp 一样，并且执行页面刷新时页面上显示的次数也在递增。但当启动另外一个浏览器执行 count_application.jsp 时，就会发现两者的区别，在新打开的浏览器中，页面上的数字不是从 1 开始重新计数，而是会接着递增下去。这是因为第一次执行 count_application.jsp 时创建了 JavaBean，而另一个浏览器执行的 count_application.jsp 仍然是属于同一个 application，所以 JavaBean 一直都存在，除非把 Web 服务器重新启动。

7.4 通过 JavaBean 访问数据库

在 JSP 中对数据库访问是一个重要的问题，在 Java 技术中，连接数据库通常是使用 JDBC 实现的。通过 JDBC 技术，JSP 程序可以访问目前流行的几乎所有的数据库，如 Oracle、SQL Server、My SQL 等。

7.4.1 连接数据库

在进行 JSP 应用程序开发时，经常需要对数据库进行查询及增删改，而这些操作使用又非常频繁。因此，可以将操作数据库的代码封装到一个 JavaBean 中。当需要更改要访问的数据库时，只要修改 JavaBean 文件即可，这样可以简化开发过程，提高代码的重用性，有利于程序的维护。

通过 JavaBean 组件连接数据库中代码的实现一般有两种方法。

1. 在类的构造方法进行初始化连接

该方法的源代码如下。

```
package bean;
import java.sql.*;
public class DBConnect {
    Connection con = null;
    //JDBC 驱动程序名称
    String drivername = "com.microsoft.jdbc.sqlserver.SQLServerDriver";
    //连接数据库 url
    String url = "jdbc:microsoft:sqlserver://localhost:1433;DatabaseName = bus";
    //连接数据库用户名为 sa
    String username = "sa";
    //连接数据库密码为空
    String password = "";
    public DBConnect() {
        try{
            //加载 JDBC 驱动程序
            Class.forName(drivername);
            //连接数据库
            con = DriverManager.getConnection(url,username,password);
            System.out.println("数据库连接成功!");
        }catch(ClassNotFoundException e){
            e.printStackTrace();
        }catch(SQLException e){
            e.printStackTrace();
        }
    }
}
```

2. 通过类中的方法进行连接数据库

该方法的源代码如下。

```
package bean;
import java.sql.*;
public class DBConnect {
    Connection con = null;
    //JDBC 驱动程序名称
    String drivername = "com.microsoft.jdbc.sqlserver.SQLServerDriver";
    //连接数据库 url
    String url = "jdbc:microsoft:sqlserver://localhost:1433;DatabaseName = bus";
    //连接数据库用户名为 sa
    String username = "sa";
    //连接数据库密码为空
    String password = "";
    public DBConnect() {
    }
    public Connection getConnection(){
        try{
            //加载 JDBC 驱动程序
            Class.forName(drivername);
            //连接数据库
            con = DriverManager.getConnection(url,username,password);
            System.out.println("数据库连接成功!");
        }catch(ClassNotFoundException e){
            e.printStackTrace();
        }catch(SQLException e){
            e.printStackTrace();
        }
        return con;
    }
}
```

7.4.2 实现对数据库的操作

通过 JavaBean 实现对数据库的信息进行查询、添加、修改、删除等操作，在 JavaBean 中操作数据库和在 JSP 页面中操作数据库是一样的，不同的是 JavaBean 只负责执行数据库操作，不关心显示方面的逻辑。这样可以有效地实现显示层和数据访问层的分离。JSP 页面仅负责数据的录入和显示，当需要对数据库进行操作时，只需调用 JavaBean 中的方法即可。

下面的代码就是通过一个 JavaBean 来说明 JavaBean 实现对数据库的几种操作方法。

```
package bean;
import java.sql.*;

public class DBUtil {
    private Connection con;
```

```
    public DBUtil() {
    }
    //查询数据库表中的信息
    public User findUser(String username){
        con = (new DBConnect()).getConnection();
        User user = null;
        Statement stmt;
        ResultSet rs;
        String sql = "select * from users where username = '" + username + "'";
        try{
            stmt = con.createStatement();
            rs = stmt.executeQuery(sql);
            while (rs.next()){
                user = new User();
                user.setUsername(rs.getString("username"));
                user.setPassword(rs.getString("password"));
            }
            rs.close();
            stmt.close();
            con.close();
        }catch(Exception e){
            e.printStackTrace();
        }
        return user;
  }
  //向数据库表中添加一个 user 的信息
  public boolean addUser(User user){
      con = (new DBConnect()).getConnection();
      PreparedStatement pstmt;
      String sql = "insert into users values(?,?)";
      boolean flag = false;
      try{
          pstmt = con.prepareStatement(sql);
          pstmt.setString(1, user.getUsername());
          pstmt.setString(2, user.getPassword());
          pstmt.execute();
          flag = true;
          pstmt.close();
          con.close();
     }catch(Exception e){
          e.printStackTrace();
     }
     return flag;
  }

  //删除数据库表中的信息
  public boolean deleteUser(String username){
      con = (new DBConnect()).getConnection();
      Statement stmt;
      String sql = "delete from users where username = '" + username + "'";
      boolean flag = false;
```

```
        try{
            stmt = con.createStatement();
            stmt.execute(sql);
            flag = true;
            stmt.close();
            con.close();
        }catch(Exception e){
            e.printStackTrace();
        }
        return flag;
    }

    //修改数据库中的 user 的信息
    public boolean modifyUser(String username,User user){
        con = (new DBConnect()).getConnection();
        PreparedStatement pstmt;
        String sql = "update users set username = ?,password = ? where username = ?";
        boolean flag = false;
        try{
            pstmt = con.prepareStatement(sql);
            pstmt.setString(1, user.getUsername());
            pstmt.setString(2, user.getPassword());
            pstmt.setString(3, username);
            pstmt.execute();
            flag = true;
            pstmt.close();
            con.close();
        }catch(Exception e){
            e.printStackTrace();
        }
        return flag;
    }
}
```

【例 7-7】 一个基于 JSP+JavaBean 的登录程序。

对创建的登录模块代码进行修改,把对数据库的操作封装在 JavaBean 中,然后在 JSP 页面中调用 JavaBean 进行登录校验。数据库 bus 中的表 users 的信息结构如图 7-8 和图 7-9 所示。本例源代码存放于本书配套素材中的相应文件中(文件路径为 code\7)。

列名	数据类型	长度	允许空
id	int	4	
username	varchar	20	✓
password	varchar	20	✓

图 7-8　表 users 的结构

id	username	password
1	admin	admin
2	chen	chen

图 7-9　表 users 的数据信息

(1) 创建一个名字为 User.java 的 JavaBean,源代码如下所示。在 User.java 中定义了 username 和 password 两个属性,用于保存表单中输入的用户名和密码。

<! -- 例程 7 - 11 User.java -->

```
package bean;
public class User {
```

```
        private String username;
        private String password;
        public String getUsername() {
            return username;
        }
        public void setUsername(String username) {
            this.username = username;
        }
        public String getPassword() {
            return password;
        }
        public void setPassword(String password) {
            this.password = password;
        }
        public User() {
        }
        public User(String password, String username) {
            this.password = password;
            this.username = username;
        }
    }
```

（2）创建一个名字为 DBConnect.java 的 JavaBean，用于连接 SQL Server 2000 数据库 bus，源代码如下所示。

<! -- 例程 7 - 12 DBConnect.java -->

```
    package bean;
    import java.sql.*;
    public class DBConnect {
        Connection con = null;
        //JDBC 驱动程序名称
        String drivername = "com.microsoft.jdbc.sqlserver.SQLServerDriver";
        //连接数据库 url
        String url = "jdbc:microsoft:sqlserver://localhost:1433;DatabaseName = buy";
        //连接数据库用户名为 sa
        String username = "sa";
        //连接数据库密码为空
        String password = "";
        public DBConnect() {
        }
        public Connection getConnection(){
            try{
                //加载 JDBC 驱动程序
                Class.forName(drivername);
                //连接数据库
                con = DriverManager.getConnection(url,username,password);
                System.out.println("数据库连接成功!");
            }catch(ClassNotFoundException e){
                e.printStackTrace();
            }catch(SQLException e){
```

```
            e.printStackTrace();
        }
        return con;
    }
}
```

(3) 创建一个用户登录 UserUtil.java 的 JavaBean，用于进行读取数据库 bus 中表 users 的用户信息，源代码如下所示。

<! -- 例程 7-13 UserUtil.java -->

```
package bean;
import java.sql.*;

public class UserUtil {
    private Connection con;
    //查询数据库表中的信息是否存在
    public boolean findUser(String username,String password){
        con = (new DBConnect()).getConnection();
        boolean flag = false;
        Statement stmt;
        ResultSet rs;
        String sql = "select * from users where username = '" + username + "' and password = '" + 
password + "'";
        try{
            stmt = con.createStatement();
            rs = stmt.executeQuery(sql);
            if (rs.next())
                flag = true;
            rs.close();
            stmt.close();
            con.close();
        }catch(Exception e){
            e.printStackTrace();
        }
        return flag;
    }
}
```

(4) 创建一个 login.html 登录页面，其源代码如下所示。

<! -- 例程 7-14 login.html -->

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>登录页面</title>

    <meta http-equiv = "keywords" content = "keyword1,keyword2,keyword3">
    <meta http-equiv = "description" content = "this is my page">
    <meta http-equiv = "content-type" content = "text/html; charset = UTF-8">

  </head>
```

```
<body>
  <div align=center>
      <form action="checklogin.jsp" method="get">
          <table border=3 cellspacing=3>
              <tr>
                  <td>用户名：</td>
                  <td><input type="text" name="username"></td>
              </tr>
              <tr>
                  <td>密 码：</td>
                  <td><input type="password" name="password"></td>
              </tr>
              <tr>
                  <td colspan="2" align="center">
                      <input type="submit" value="提交">
                      <input type="reset" value="重置">
                  </td>
              </tr>
            </table>
          </form>
      </div>
    </table>
  </body>
</html>
```

该页面显示效果如图 7-10 所示。

(5) 创建一个 checklogin.jsp，用于获取 login.html 页面中在文本框输入的用户名和密码，并调用 UserUtil 的实例，判断用户名和密码是否正确，并显示检验信息，其源代码如下所示。



图 7-10　login.html 设计界面

```
<!-- 例程 7-15 checklogin.jsp -->

<%@ page language="java" pageEncoding="gb2312"%>
<%@page import="bean.LoginBean"%>
<jsp:useBean id="user" class="bean.LoginBean" scope="page">
<jsp:setProperty name="user" property="*"/></jsp:useBean>
<%
    if (user.checklogin()){
        String username = request.getParameter("username");
        session.setAttribute("username", username);
 %>
        <jsp:forward page="welcome.jsp"></jsp:forward>
<%
    } else {
 %>
        <jsp:forward page="error.jsp"></jsp:forward>
<%
    }
 %>
```

7.5　上机指导

7.5.1　猜数字游戏

1. 练习目标

(1) 熟练掌握 JavaBean 组件的创建。
(2) 熟练掌握在 JSP 页面中使用 JavaBean 组件。
(3) 熟悉 JSP+JavaBean 程序的调试运行。

2. 练习指导

编写一个猜蛋糕价格的游戏程序，具体操作过程如下。本例源代码存放于本书配套素材中的相应文件中(文件路径为 code\上机指导\7)。

(1) 新建一个名称为 GuessGame 的类，GuessGame.java 源代码如下。

<! -- 例程 7-16 GuessGame.java -->

```
package bean;
import java.util.*;
public class GuessGame {
    private int answer;
    private int guess;
    private boolean success;
    private String info;
    private int count;
    public GuessGame() {
        reset();//重置开始
    }
    //设置和调用成员属性,完成游戏功能
    public void setGuess(String guess) {
        count++;
        try {
            this.guess = Integer.parseInt(guess);
        }
        catch (NumberFormatException e) {
            this.guess = -1;
        }
        //判断所输入的数字与实际价格是否相同,或输入数字是否符合要求
        if (this.guess == answer) {
            success = true;
        }
        else if (this.guess == -1) {
            info = "出错,再猜一次!";
        }
        else if (this.guess < answer) {
```

```
                info = "您猜的价格小了!";
            }
            else if (this.guess > answer) {
                info = "您猜的价格大了!";
            }
            //输入数字
            if(this.guess > 1000){
                info = "请输入 1 到 1000 之间的数字!";
            }
        }
        public boolean getSuccess() {
            return success;
        }
        public String getInfo() {
            return info;
        }
        public int getCounter() {
            return count;
        }
        public int getAnswer(){
            return answer;
        }
        public void reset() {
            //产生随机数,控制在 1～1000 之间
            answer = Math.abs(new Random().nextInt() % 1000) + 1;
            success = false;
            count = 0;
        }
}
```

（2）新建 JSP 页面用于显示和输入蛋糕价格信息，guessgame.jsp 页面源代码如下。

<! -- 例程 7-17 guessgame.jsp -->

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<%
String path = request.getContextPath();
String basePath = request.getScheme() + "://" + request.getServerName() + ":" + request.
getServerPort() + path + "/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
   <head>
      <base href="<%=basePath%>">
      <title>猜蛋糕价格游戏</title>
      <meta http-equiv="pragma" content="no-cache">
      <meta http-equiv="cache-control" content="no-cache">
      <meta http-equiv="expires" content="0">
      <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
      <meta http-equiv="description" content="This is my page">
  </head>
```

```
<body>
    <jsp:useBean id = "game" class = "bean.GuessGame" scope = "session" />
    <jsp:setProperty name = "game" property = " * " />
    <%
          if(game.getCounter() == 0){
   %>
  <center>
  <font size = "4" color = "red">猜价格</font>
  <hr>
  <p align = "center">
      <img src = "images/cake.jpg" alt = "蛋糕" width = "139px" height = "104px" border
       = 3>
  </p>
  <form method = "get" name = "form1">
      <b>请输入价格: </b>
      <input type = "text" name = "guess">
      <input type = "submit" value = "确定" name = "submit">
  </form>
  </center>
  <%
      }
      else if(game.getSuccess())
      {
   %>
  <center>
      <b>猜对了,它归你了。你猜了 <% = game.getCounter() %> 次。</b>
      <br>
      <a href = "guessgame.jsp">再来一次?</a></center>
  <%
          game.reset();
      }
      else{
   %>
  <center><b>继续努力,<% = game.getInfo() %>.
  你已经猜了 <% = game.getCounter() %> 次.
  <br>
  输入你猜的价格:
  <form method = "get" name = "form2">
       <input type = "text" name = "guess">
       <input type = "submit" value = 确定 name = "submit">
  </form>
  </b></center>
  <%
      }
   %>
</body>
</html>
```

(3) 调试程序,运行程序。

7.5.2 简单的购物程序

1. 练习目标

(1) 熟练掌握JSP页面及JavaBean类的创建。
(2) 熟练掌握JavaBean组件对数据库访问。

2. 练习指导

编写一个网上购物程序,该程序包括用户注册、登录、购物车、商品列表分页显示等功能。其操作过程如下。本例源代码存放于本书配套素材中的相应文件中(文件路径为code\上机指导\7)。

(1) 利用SQL Server 2000创建数据库buy,数据库中包括两张表books和users,分别用于存放图书信息和用户信息,表结构如图7-11和图7-12所示。

列名	数据类型	长度	允许空
id	int	4	
bookname	varchar	50	✓
author	varchar	30	✓
publish	varchar	80	✓
isbn	varchar	20	✓
content	varchar	4000	✓
price	float	8	✓
pubdate	datetime	8	✓
amount	int	4	✓
leavecnt	int	4	✓

图7-11 表books的数据结构

列名	数据类型	长度	允许空
username	varchar	20	
password	varchar	20	✓
name	varchar	30	✓
sex	char	1	✓
address	varchar	100	✓
pcode	varchar	6	✓
email	varchar	30	✓
tel	varchar	20	✓
regdate	datetime	8	✓

图7-12 表users的数据结构

(2) 网上购物所用到的主要程序页面代码文件及功能如表7-1所示。

表7-1 程序文件及功能

序号	文件名	类型	功能描述
1	Login1.html	页面	用户登录录入页面
2	Login1.jsp	页面	检测用户登录
	logout.jsp	页面	用户注销
3	reg.html	页面	用户信息注册录入页面
4	reg.jsp	页面	写入用户注册信息
5	booklist.jsp	页面	列表分页显示所有图书
6	purchase.jsp	页面	录入某本书购买数量
7	cartlist.jsp	页面	购物车页面,显示已加入到购物车中的图书信息
8	DBConnect.java	JavaBean	连接数据库
9	BookUtil.java	JavaBean	显示图书查询分页显示功能
10	Cart.java	JavaBean	实现购物车操作
11	User.java	JavaBean	封装用户信息
12	UserUtil.java	JavaBean	用户信息添加及查找用户功能
13	Book.java	JavaBean	封装图书信息
14	BookUtil.java	JavaBean	图书信息分页显示及查找图书信息功能
15	Item.java	JavaBean	封装购物车的一个购买的条目信息

（3）数据库连接 JavaBean 代码。

<! -- 例程 7-18 DBConnect.java -->

```
package bean;
import java.sql.*;
public class DBConnect {
    Connection con = null;
    //JDBC 驱动程序名称
    String drivername = "com.microsoft.jdbc.sqlserver.SQLServerDriver";
    //连接数据库 url
    String url = "jdbc:microsoft:sqlserver://localhost:1433;DatabaseName = buy";
    //连接数据库用户名为 sa
    String username = "sa";
    //连接数据库密码为空
    String password = "";
    public DBConnect() {
    }
    public Connection getConnection(){
        try{
            //加载 JDBC 驱动程序
            Class.forName(drivername);
            //连接数据库
            con = DriverManager.getConnection(url,username,password);
            }catch(ClassNotFoundException e){
            e.printStackTrace();
        }catch(SQLException e){
            e.printStackTrace();
        }
        return con;
    }
    public static void main(String args[]){
        (new DBConnect()).getConnection();
    }
}
```

（4）用户相关模块所使用的 JavaBean 组件代码。

<! -- 例程 7-19 User 代码 -->

```
package bean;
import java.util.Date;
public class User {
    private String username;
    private String password;
    private String name;
    private char sex;
    private String address;
    private String pcode;
    private String email;
    private String tel;
```

```
    private Date regdate;
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public char getSex() {
        return sex;
    }
    public void setSex(char sex) {
        this.sex = sex;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    public String getPcode() {
        return pcode;
    }
    public void setPcode(String pcode) {
        this.pcode = pcode;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getTel() {
        return tel;
    }
    public void setTel(String tel) {
        this.tel = tel;
    }
    public Date getRegdate() {
        return regdate;
```

```
    }
    public void setRegdate(Date regdate) {
        this.regdate = regdate;
    }
    public User() {
    }
    public User(String address, String email, String name, String password,
            String pcode, Date regdate, char sex, String tel, String username) {
        super();
        this.address = address;
        this.email = email;
        this.name = name;
        this.password = password;
        this.pcode = pcode;
        this.regdate = regdate;
        this.sex = sex;
        this.tel = tel;
        this.username = username;
    }
}
```

<! -- 例程 7-20 UserUtil.java -->

```
package bean;
import java.sql.*;

public class UserUtil {
    private Connection con;
    //查询数据库表中的用户信息是否存在
    public boolean findUser(String username,String password){
        con = (new DBConnect()).getConnection();
        boolean flag = false;
        Statement stmt;
        ResultSet rs;
        String sql = "select * from users where username = '" + username + "' and password = '" +
password + "'";
        try{
            stmt = con.createStatement();
            rs = stmt.executeQuery(sql);
            if (rs.next())
                flag = true;
            rs.close();
            stmt.close();
            con.close();
        }catch(Exception e){
            e.printStackTrace();
        }
        return flag;
    }
    //向数据库表中添加一个 user 用户信息
    public boolean addUser(User user){
```

```
        boolean flag = false;
        if (user != null){
            con = (new DBConnect()).getConnection();
            Statement stmt;
            try{
                stmt = con.createStatement();
                String sql = "select * from users where username = '" + user.getUsername() +"'";
                ResultSet rs = stmt.executeQuery(sql);
                if (rs.next()){
                    rs.close();
                    return flag;
                }
                sql = "insert into users(username, password, name, sex, address, tel, pcode,
email,regDate) values ('" + user.getUsername() + "','" + user.getPassword() + "','" + user.getName
() + "','" + user.getSex() + "','" + user.getAddress() + "','" + user.getTel() + "','" + user.getPcode()
+ "','" + user.getEmail() + "',getdate())";
                stmt.execute(sql);
                flag = true;
                stmt.close();
                con.close();
            }catch(Exception e){
                e.printStackTrace();
            }
        }
        return flag;
    }
```

（5）用户登录模块实现的源代码如下。

<!-- 例程 7-21 login1.html -->

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>用户登录</title>
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="this is my page">
    <meta http-equiv="content-type" content="text/html; charset=gb2312">
    <script language="javascript">
function checkform() {
    if (document.form1.username.value == "" || document.form1.password.value == ""){
        alert("用户名或密码为空!");
        return false;
    }
    return true;
  }
</script>
  </head>
  <body>
```

```
<form method="post" action="login.jsp" name="form1" onSubmit="return checkform();">
    <div align="center"></div>
        <table align="center" border="1" cellspacing="1" cellpadding="1" bgcolor="#CCCCCC">
            <tr>
                <td colspan="2" align="center"><font face="黑体" size="4">用户登录
</font></td>
            </tr>
            <tr>
                 <td>用户名:</td>
                 <td>
                    <input type="text" name="username">
                </td>
            </tr>
            <tr>
                <td>密 码:</td>
                <td>
                    <input type="password" name="password">
                </td>
            </tr>
            <tr>
                <td align="center" colspan="2">
                    <input type="submit" name="submit" value="登录"> 
                    <input type="reset" name="reset" value="重置"> 
                    <input type="Button" name="reg" value="注册"
onClick="javascript:self.location='reg.html'">
                </td>
            </tr>
  </table>
  </form>
  </body>
</html>
```

<!-- 例程 7-22 login1.jsp -->

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.
getServerPort()+path+"/";
 %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
      <base href="<%=basePath%>">

      <title>用户登录检测</title>

      <meta http-equiv="pragma" content="no-cache">
      <meta http-equiv="cache-control" content="no-cache">
      <meta http-equiv="expires" content="0">
      <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
```

```
        <meta http-equiv="description" content="This is my page">
    </head>
<jsp:useBean id="userutil" scope="page" class="bean.UserUtil" />
<%
        String msg = "";
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        if (userutil.findUser(username,password)){
            session.setAttribute("username",username);
            response.sendRedirect("booklist.jsp");
        }
        else{
            msg = "登录出错!";
        }
%>
   <body>
<div align="center">
<% = msg %> <a href="login.html">重新登录</a>
</div>
  </body>
</html>
```

(6) 用户注册模块实现代码。

```
                    <!-- 例程 7-23 reg.html -->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
   <head>
      <title>用户注册</title>
      <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
      <meta http-equiv="description" content="this is my page">
      <meta http-equiv="content-type" content="text/html; charset=gb2312">
function openScript(url,name, width, height){
      var Win = window.open(url,name,'width=' + width + ',height=' + height + ',resizable=1,
scrollbars=yes,menubar=no,status=yes');
}
function checkform() {
      if (document.form1.username.value==""){
         alert("用户名不能为空");
         document.form1.username.focus();
         return false;
     }
     if (document.form1.passwd.value==""){
         alert("用户密码不能为空");
         document.form1.passwd.focus();
         return false;
     }
     if (document.form1.passwd.value != document.form1.passconfirm.value){
         alert("确认密码不相符!");
         document.form1.passconfirm.focus();
```

```
            return false;
        }
        return true;
    }
</script>
  </head>
  <body>
    <div align = "center">
  <p>网上书店用户注册</p>
  <form name = "form1" method = "post" action = "reg.jsp" onSubmit = "return checkform();">
    <table width = "400" border = "1" cellspacing = "1" cellpadding = "1" bgcolor = "#CCCCCC">
      <tr>
        <td colspan = "2" align = "center"><font face = "黑体" size = "4">用户注册</font>
        </td>
      </tr>
      <tr>
        <td width = "120" align = "right">用户名:</td>
        <td width = "280">
           <input type = "text" name = "username" maxlength = "20" size = "20" >
        </td>
      </tr>
      <tr>
        <td width = "120" align = "right">密   码:</td>
        <td width = "280">
           <input type = "password" name = "password" maxlength = "20" size = "20">
        </td>
      </tr>
      <tr>
        <td width = "120" align = "right">确认密码:</td>
        <td width = "280">
           <input type = "password" name = "passconfirm" maxlength = "20" size = "20">
        </td>
      </tr>
      <tr>
        <td width = "120" align = "right">真实姓名:</td>
        <td width = "280">
           <input type = "text" name = "name" maxlength = "20" size = "20">
        </td>
      </tr>
      <tr>
        <td width = "120" align = "right">性   别:</td>
        <td width = "280">
           <input type = "radio" checked = "checked" name = "sex" value = "0">男
           <input type = "radio" name = "sex" value = "1">女
        </td>
      </tr>
      <tr>
        <td width = "120" align = "right">电子邮件:</td>
        <td width = "280">
           <input type = "text" name = "email" maxlength = "50" size = "25">
        </td>
```

```
        </tr>
        <tr>
          <td width="120" align="right">联系电话:</td>
          <td width="280">
              <input type="text" name="phone" maxlength="25" size="16">
          </td>
        </tr>
        <tr>
          <td width="120" align="right">联系地址:</td>
          <td width="280">
              <input type="text" name="address" maxlength="150" size="40">
          </td>
        </tr>
        <tr>
          <td width="120" align="right">邮政编码:</td>
          <td width="280">
              <input type="text" name="pcode" maxlength="6" size="6">
          </td>
        </tr>
        <tr>
          <td colspan="2" align="center">
              <input type="submit" name="Submit" value="注册">  
              <input type="reset" name="reset" value="取消">
          </td>
        </tr>
      </table>
    </form>
    <p> </p>
</div>
    </body>
</html>
```

```
<!-- 例程 7-24 reg.jsp -->
```

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<%
String path = request.getContextPath();
String basePath = request.getScheme() + "://" + request.getServerName() + ":" + request.
getServerPort() + path + "/";
 %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
   <head>
      <base href="<%=basePath%>">
      <title>用户注册</title>
      <meta http-equiv="pragma" content="no-cache">
      <meta http-equiv="cache-control" content="no-cache">
      <meta http-equiv="expires" content="0">
      <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
      <meta http-equiv="description" content="This is my page">
   </head>
```

```
<%
      request.setCharacterEncoding("GB2312");
 %>
<%@ page import="bean.User"%>
<jsp:useBean id="userUtil" scope="page" class="bean.UserUtil"/>
   <body>
<%
      String msg = "";
      User user = new User();
      String username = request.getParameter("username");
      String password = request.getParameter("password");
      String name = request.getParameter("name");
      String sex = request.getParameter("sex");
      String address = request.getParameter("address");
      String pcode = request.getParameter("pcode");
      String tel = request.getParameter("tel");
      String email = request.getParameter("email");
      user.setUsername(username);
      user.setPassword(password);
      user.setName(name);
      user.setSex(sex.charAt(0));
      user.setAddress(address);
      user.setPcode(pcode);
      user.setTel(tel);
      user.setEmail(email);
      if(userUtil.addUser(user)){
          session.setAttribute("username",user.getUsername());
          response.sendRedirect("booklist.jsp");
    }else{
          msg = "注册时出现错误,请稍后再试";
    }
 %>
    <%=msg%>
  </body>
</html>
```

<!-- 例程 7-25 logout.jsp -->

```
<%
    session.invalidate();
    response.sendRedirect("login.html");
 %>
```

（7）购物模块使用的相关 JavaBean 组件代码。

<!-- 例程 7-26 Book.java -->

```
package bean;
import java.util.Date;
```

```
public class Book {
    private int id;
    private String bookname;
    private String author;
    private String publish;
    private String isbn;
    private String content;
    private float price;
    private Date pubdate;
    private int amount;
    private int leavecnt;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getBookname() {
        return bookname;
    }
    public void setBookname(String bookname) {
        this.bookname = bookname;
    }
    public String getAuthor() {
        return author;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
    public String getPublish() {
        return publish;
    }
    public void setPublish(String publish) {
        this.publish = publish;
    }
    public String getIsbn() {
        return isbn;
    }
    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }
    public String getContent() {
        return content;
    }
    public void setContent(String content) {
        this.content = content;
    }
    public float getPrice() {
        return price;
    }
    public void setPrice(float price) {
```

```
        this.price = price;
    }
    public Date getPubdate() {
        return pubdate;
    }
    public void setPubdate(Date pubdate) {
        this.pubdate = pubdate;
    }
    public int getAmount() {
        return amount;
    }
    public void setAmount(int amount) {
        this.amount = amount;
    }
    public int getLeavecnt() {
        return leavecnt;
    }
    public void setLeavecnt(int leavecnt) {
        this.leavecnt = leavecnt;
    }
    public Book(int amount, String author, String bookname, String content,
            int id, String isbn, int leavecnt, float price, Date pubdate,
            String publish) {
        this.amount = amount;
        this.author = author;
        this.bookname = bookname;
        this.content = content;
        this.id = id;
        this.isbn = isbn;
        this.leavecnt = leavecnt;
        this.price = price;
        this.pubdate = pubdate;
        this.publish = publish;
    }
    public Book() {
    }
}
```

<! -- 例程 7-27 BookUtil.java -->

```
package bean;
import java.sql.*;
import java.util.Vector;
public class BookUtil {
    private Vector booklist;            //显示图书列表向量数组
    private int page = 1;               //显示的页码
    private int pageSize=10;            //每页显示的图书数
    private int pageCount =0;           //页面总数
    private long recordCount =0;        //查询的记录总数
    private Connection con;
    private Statement stmt;
```

```
    private ResultSet rs;
    public BookUtil() {
    }
    public Vector getBooklist() {
        return booklist;
    }
    public void setBooklist(Vector booklist) {
        this.booklist = booklist;
    }
    public int getPage() {
        return page;
    }
    public void setPage(int page) {
        this.page = page;
    }
    public int getPageSize() {
        return pageSize;
    }
    public void setPageSize(int pageSize) {
        this.pageSize = pageSize;
    }
    public int getPageCount() {
        return pageCount;
    }
    public void setPageCount(int pageCount) {
        this.pageCount = pageCount;
    }
    public long getRecordCount() {
        return recordCount;
    }
    public void setRecordCount(long recordCount) {
        this.recordCount = recordCount;
    }
    //查询参数 Page 所指定页的记录,并存放在向量数组 booklist 中
    public boolean execute(int page){
        //取出记录数
        String sql = "select count( * ) from books";
        try{
            con = (new DBConnect()).getConnection();
            stmt = con.createStatement();
            rs = stmt.executeQuery(sql);
            if (rs.next()) recordCount = rs.getInt(1);
            rs.close();
            stmt.close();
            con.close();
        }
        catch (Exception e){
            return false;
        }
        //设定有多少 pageCount
        if (recordCount < 1)
```

```
        pageCount = 0;
    else
        pageCount = (int)(recordCount - 1) / pageSize + 1;
    //检查查看的页面数是否在范围内
    if (page < 1)
        page = 1;
    else if (page > pageCount)
        page = pageCount;
    //当前页显示 cnt 条记录
    int cnt;
    if (page == pageCount)
        cnt = (int)recordCount % pageSize;
    else
        cnt = pageSize;
    //当前页的第一条记录号
    int recordno = (page - 1) * pageSize + 1;
    //sql 为倒序取值
    sql = "select * from books order by id";
    try{
        con = (new DBConnect()).getConnection();
        stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_
READ_ONLY);
        rs = stmt.executeQuery(sql);
        rs.absolute(recordno);
        booklist = new Vector();
        do
        {
            Book book = new Book();
            book.setId(rs.getInt("id"));
            book.setBookname(rs.getString("bookname"));
            book.setAuthor(rs.getString("author"));
            book.setPublish(rs.getString("publish"));
            book.setIsbn(rs.getString("isbn"));
            book.setContent(rs.getString("content"));
            book.setPrice(rs.getFloat("price"));
            book.setAmount(rs.getInt("amount"));
            book.setLeavecnt(rs.getInt("leavecnt"));
            book.setPubdate(rs.getDate("pubdate"));
            booklist.add(book);
            cnt--;
        }while (rs.next() && cnt != 0);
        rs.close();
        stmt.close();
        con.close();
        return true;
    }
    catch (SQLException e){
        return false;
    }
}
//查询参数 newid 所指定的图书,并返回查询结果
```

```
    public Vector getOnebook(int newid ){
        try
        {
            con = (new DBConnect()).getConnection();
            stmt = con.createStatement();
            String sql = "select * from books where id = " + newid;
            rs = stmt.executeQuery(sql);
            if (rs.next())
            {   booklist = new Vector();
                Book book = new Book();
                book.setId(rs.getInt("id"));
                book.setBookname(rs.getString("bookname"));
                book.setAuthor(rs.getString("author"));
                book.setPublish(rs.getString("publish"));
                book.setIsbn(rs.getString("isbn"));
                book.setContent(rs.getString("content"));
                book.setPrice(rs.getFloat("price"));
                book.setAmount(rs.getInt("amount"));
                book.setLeavecnt(rs.getInt("leavecnt"));
                book.setPubdate(rs.getDate("pubdate"));
                booklist.addElement(book);
            }
            rs.close();
            stmt.close();
            con.close();
            return booklist;
        }
        catch (Exception e){
            return null;
        }
    }
}
```

<! -- 例程 7 - 28 Cart.java -->

```
package bean;
import java.sql.*;
import java.util.Vector;
import javax.servlet.http.*;
public class Cart {
    private HttpServletRequest request;        //建立页面请求
    private HttpSession session;               //页面的 session
    private Vector purchaselist;               //显示图书列表向量数组
    private boolean isEmpty = false;           //库中的书数量是否达到购买的数
    private int leaveBook = 0;                 //库存数量
    private Connection con;
    private Statement stmt;
    private ResultSet rs;
    public Cart() throws Exception{
        super();
    }
```

```
public Vector getPurchaselist() {
    return purchaselist;
}
public void setIsEmpty(boolean flag){
    isEmpty = flag;
}
public boolean getIsEmpty() {
    return isEmpty;
}
public void setLeaveBook(int bknum) {
    leaveBook = bknum;
}
public int getLeaveBook() {
    return leaveBook;
}
public boolean addnew(String id,String num,HttpServletRequest newrequest)throws Exception{
    request = newrequest;
    long bookid = 0;
    int amount = 0;
    try
    {
        bookid = Long.parseLong(id);
        amount = Integer.parseInt(num);
    }
    catch (Exception e)
    {
        return false;
    }
    if (amount<1) return false;
    session = request.getSession(false);
    if (session == null)
    {
        return false;
    }
    purchaselist = (Vector)session.getAttribute("cart");
    String sql = "select leavecnt from books where id=" + bookid;
    try{
        con=(new DBConnect()).getConnection();
        stmt = con.createStatement();
        rs = stmt.executeQuery(sql);
        if (rs.next()){
            if (amount>rs.getInt(1)){
                leaveBook = rs.getInt(1);
                isEmpty = true;
                return false;
            }
        }
    }
    catch (SQLException e){
        return false;
    }
```

```
        Item iList = new Item();
        iList.setBookNo(bookid);
        iList.setAmount(amount);
        boolean match = false;                    //是否购买过该图书
        if (purchaselist == null){                //第一次购买
            purchaselist = new Vector();
            purchaselist.addElement(iList);
        }
        else{                                     // 不是第一次购买
            for (int i = 0; i < purchaselist.size(); i++ ) {
                Item itList = (Item) purchaselist.elementAt(i);
                if ( iList.getBookNo() == itList.getBookNo() ) {
                    itList.setAmount(itList.getAmount() + iList.getAmount());
                    purchaselist.setElementAt(itList,i);
                    match = true;
                    break;
                }                                 //if name matches 结束
            }                                     // for 循环结束
            if (!match) {
                purchaselist.addElement(iList);
            }
        }
        session.setAttribute("cart", purchaselist);
        return true;
    }
    public boolean modiShoper(String id, String num, HttpServletRequest newrequest) throws
Exception {
        request = newrequest;
        long bookid = 0;
        int amount = 0;
        try{
            bookid = Long.parseLong(id);
            amount = Integer.parseInt(num);
        }
        catch (Exception e){
            return false;
        }
        if (amount < 1) return false;
        session = request.getSession(false);
        if (session == null){
            return false;
        }
        purchaselist = (Vector)session.getAttribute("cart");
        if (purchaselist == null){
            return false;
        }
        String sql = "select leavecnt from books where id = " + bookid;
        try
        {
            con = (new DBConnect()).getConnection();
            stmt = con.createStatement();
```

```
            rs = stmt.executeQuery(sql);
            if (rs.next()){
                if (amount > rs.getInt(1)){
                    leaveBook = rs.getInt(1);
                    isEmpty = true;
                    return false;
                }
            }
        }
        catch (SQLException e){
            return false;
        }
        for (int i = 0; i < purchaselist.size(); i++ ){
            Item itList = (Item) purchaselist.elementAt(i);
            if ( bookid == itList.getBookNo() ){
                itList.setAmount(amount);
                purchaselist.setElementAt(itList,i);
                break;
            }
        }
        return true;
    }
    public boolean delShoper(String delID,HttpServletRequest newrequest)throws Exception {
        request = newrequest;
        long bookid = 0;
        try{
            bookid = Long.parseLong(delID);
        }
        catch (Exception e){
            return false;
        }
        session = request.getSession(false);
        if (session == null){
            return false;
        }
        purchaselist = (Vector)session.getAttribute("cart");
        if (purchaselist == null){
            return false;
        }
        for (int i = 0; i < purchaselist.size(); i++ ) {
            Item itList = (Item) purchaselist.elementAt(i);
            if ( bookid == itList.getBookNo() ) {
                purchaselist.removeElementAt(i);
                break;
            }
        }
        return true;
    }
}
```

<! -- 例程 7-29 Item.java -->

```
package bean;

public class Item {
    private long id;          //ID 序列号
    private long bookNo;      //图书表序列号
    private int amount;       //订货数量
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }
    public long getBookNo() {
        return bookNo;
    }
    public void setBookNo(long bookNo) {
        this.bookNo = bookNo;
    }
    public int getAmount() {
        return amount;
    }
    public void setAmount(int amount) {
        this.amount = amount;
    }
    public Item() {
        this.id = 0;
        this.bookNo = 0;
        this.amount = 0;
    }
}
```

（8）购物模块实现代码。

<! -- 例程 7-30 booklist.jsp -->

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<%@ page import="bean.Book"%>
<%
        response.setCharacterEncoding("gb2312");
        if(session.getAttribute("username")==null){
            response.sendRedirect("login.jsp");
        }
%>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.
getServerPort()+path+"/";
%>
```

```
<! DOCTYPE HTML PUBLIC " - //W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href = "<% = basePath %>">

    <title>网上书店 -- 选购图书</title>

    <meta http - equiv = "pragma" content = "no - cache">
    <meta http - equiv = "cache - control" content = "no - cache">
    <meta http - equiv = "expires" content = "0">
    <meta http - equiv = "keywords" content = "keyword1,keyword2,keyword3">
    <meta http - equiv = "description" content = "This is my page">
    <script language = "javascript">
</script>
  <style type = "text/css">
  </style>
  </head>
<jsp:useBean id = "bookutil" scope = "page" class = "bean.BookUtil" />
<%
     String requestpage  = "";
     if (request.getParameter("page") == null || request.getParameter("page").equals("")) {
        requestpage = "1";
     }else{
        requestpage  =  request.getParameter("page");
     }
     int pageno  =  Integer.parseInt(requestpage);
     bookutil.setPage(pageno);
%>
  <body>
    <div align = "center">
    <table width = "800" border = "0" cellspacing = "1" cellpadding = "1">
    <tr>
        <td width = "100">  </td>
        <td width = "150"><a href = "booklist.jsp">在线购物</a></td>
        <td width = "150"><a href = "cartlist.jsp">我的购物车</a></td>
        <td width = "150">当前用户：<% = (String)session.getAttribute("username") %></td>
        <td width = "250"><a href = "logout.jsp">用户注销</a></td>
    </tr>
    <tr>
        <td colspan = "3"></td>
    </tr>
    </table>

    <table width = "600" border = "0" cellspacing = "1" cellpadding = "1">
    <tr valign = "center">
        <td height = "40" align = "center"><span class = "booktitle">网上书店图书列表
</span></td>
    </tr>
    <tr>
        <td>
```

```
            <table width="100%" border="0" cellspacing="0" cellpadding="0">
<%
    if (bookutil.execute(pageno)) {
        if (bookutil.getBooklist().size()>0 ){
            for (int i=0;i<bookutil.getBooklist().size();i++ ){
                Book book = (Book) bookutil.getBooklist().elementAt(i);
%>
            <tr>
                <td width="100" height="30">图书名称:</td>
                <td><span class="bookname_style"><% = book.getBookname() %></span></td>
            </tr>
            <tr>
                <td height="20">ISBN:</td>
                <td height="20"><% = book.getIsbn() %></td>
            </tr>
            <tr>
                <td height="20">作者:</td>
                <td height="20"><% = book.getAuthor() %></td>
            </tr>
            <tr>
                <td height="20">出版社:</td>
                <td height="20"><% = book.getPublish() %></td>
            </tr>
            <tr>
                <td height="20">单价:</td>
                <td height="20">¥<% = book.getPrice()%>元</td>
            </tr>
            <tr>
                <td height="20">出版时间:</td>
                <td height="20"><% = book.getPubdate()%></td>
            </tr>
            <tr>
               <td colspan="2" align="right"><a href="javascript:openScript('purchase.
jsp?bookid=<% = book.getId() %>','pur',300,250)">我要购买</a></td>
            </tr>
            <tr>
                <td colspan="2"><hr></td>
            </tr>
<%      }
    }else {
        out.println("<tr><td align='center' colspan=6> 暂时没有此类图书资料</td>
</tr>");
    }
} else {
%>
            <tr>
                <td align="center" colspan=6> 数据库出错,请稍后</td>
            </tr>
<% } %>
        </table>
每页 10 条信息,共<% = bookutil.getRecordCount() %>条 第<% = bookutil.getPage() %>页 共
```

```
<% =bookutil.getPageCount()%>页
<br>
    <table>
    <tr>
        <td>
<%
    if(bookutil.getPage()==1){
        out.print(" 首页 上一页");
    }else{
 %>
        <a href="booklist.jsp?page=1">首页</A>
        <a href="booklist.jsp?page=<% = bookutil.getPage()-1 %>">上一页</A>
<%
    }
 %>
<%
    if(bookutil.getPage()>=bookutil.getPageCount()){
        out.print("下一页 尾页");
    }else{
 %>
        <a href="booklist.jsp?page=<% = bookutil.getPage()+1%>">下一页</A>
        <a href="booklist.jsp?page=<% = bookutil.getPageCount()%>">尾页</A>
<%
    }
 %>
转到第   <select name="pageno" onChange="javascript:gopage()">
<%
    for(int i=1;i<=bookutil.getPageCount();i++){
        if (i== bookutil.getPage()){
 %>
    <option selected value=<% =i%>><% =i%></option>
<%
        }else{
 %>
            <option value=<% =i%>><% =i%></option>
<%
        }
    }
 %>
    </select>页
        </td>
    </tr>
    </table>
      </td>
    </tr>
    </table>
    </div>
</body>
</html>
```

```
<!-- 例程 7-31 purchase.jsp -->

<%@ page language="java" import="java.util.*" pageEncoding="GB18030"%>
<%@ page import="bean.Book" %>
<%
    if(session.getAttribute("username")==null){
        response.sendRedirect("login.jsp");
    }
%>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.
getServerPort()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">

    <title>网上书店--购买图书</title>

    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
    <script language="javascript">

function openScript(url,name, width, height)
{
    var Win = window.open(url,name,'width=' + width + ',height=' + height + ',resizable=
1,scrollbars=yes,menubar=no,status=yes' );
}

function check()
{
    if (document.form1.amount.value<1){
        alert("你的购买数量有问题");
        document.form1.amount.focus();
        return false;
    }
    return true;
}
</script>
  </head>

<jsp:useBean id="bookutil" scope="page" class="bean.BookUtil" />
<jsp:useBean id="cart" scope="page" class="bean.Cart" />

<%
    String msg = "";
    String submits = request.getParameter("Submit");
```

```
        int id = 0;
        if (submits! = null && !submits.equals("")){
            String bookid  =  request.getParameter("bookid");
            String amount  =  request.getParameter("amount");
            if (cart.addnew(bookid,amount,request)){
                msg  =  "你需要的图书已经放入你的购物车中!";
            }else if (cart.getIsEmpty()){
                msg  =  "库存图书数量不足!只剩" + cart.getLeaveBook() + "本";
            }else {
                msg  =  "暂时不能购买!";
            }
        }else {
            if (request.getParameter("bookid") == null || request.getParameter("bookid").equals("")) {
                msg  =  "你购买的图书不存在!";
            } else {
                try {
                    id  =  Integer.parseInt(request.getParameter("bookid"));
                    if (bookutil.getOnebook(id) == null){
                        msg  =  "你要购买的图书不存在!";
                    }
                } catch (Exception e){
                    msg  =  "你要购买的图书不存在!";
                }
            }
        }
    %>
     <body onload = "javascript:window.focus();">
    <div align = "center">
     <p><br></p><p>网上书店欢迎你选购图书!</p>
      <% if(!msg.equals("")){
            out.println(msg);
        } else {
            Book bk  =  (Book) bookutil.getBooklist().elementAt(0);
      %>
     <table width = "90 % " border = "0" cellspacing = "2" cellpadding = "1">
       <form name = "form1" method = "post" action = "purchase.jsp">
         <tr>
           <td align = "center">图书名:<% = bk.getBookname() %></td>
         </tr>
         <tr align = "center">
           <td>你想要的数量:
             <input type = "text" name = "amount" maxlength = "4" size = "3" value = "1"> 本</td>
         </tr>
         <tr align = "center">
           <td>
             <input type = "hidden" name = "bookid" value = "<% = id %>">
             <input type = "submit" name = "Submit" value = "购 买" onclick = "return(check());">
             <input type = "reset" name = "Reset" value = "取 消">
           </td>
         </tr>
       </form>
```

```
  </table>
<% } %>
  <br>
  <p><a href="javascript:window.close()">关闭窗口</a></p>
</div>
  </body>
</html>
```

<!-- 例程 7-32 cartlist.jsp -->

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<%@ page import="bean.*" %>
<%
        if(session.getAttribute("username")==null){
            response.sendRedirect("login.html");
        }
 %>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.
getServerPort()+path+"/";
 %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">

    <title>网上书店--我的购物车</title>

    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
      <style type="text/css">
  </style>
<script language="javascript">
function openScript(url,name, width, height){
     var Win = window.open(url,name,'width=' + width + ',height=' + height + ',resizable=
     1,scrollbars=yes,menubar=no,status=yes');
}
function checklogin() {
     if (document.payout.userid.value=="")
     {
         alert("你还没有登录,请登录后再提交购物清单。");
         return false;
     }
     return true;
}
function check()
```

```
{
    if (document.change.amount.value < 1){
        alert("你的购买数量有问题");
        document.change.amount.focus();
        return false;
    }
    return true;
}
</script>
  </head>
<jsp:useBean id = "bookutil" scope = "page" class = "bean.BookUtil" />
<jsp:useBean id = "cart" scope = "page" class = "bean.Cart" />

<%
    String userid = (String) session.getAttribute("userid");
    if ( userid == null )
        userid = "";
    String modi = request.getParameter("modi");
    String del = request.getParameter("del");
    String clearCar = request.getParameter("clear");
    String msg = "";
    if (modi! = null && !modi.equals("")) {
        String id = request.getParameter("bookid");
        String amount = request.getParameter("amount");
        if (!cart.modiShoper(id,amount,request) ){
            if (cart.getIsEmpty())
                msg = "你要修改购买的图书数量不足你购买的数量!";
            else
                msg = "修改购买数量出错!";
        } else {
            msg = "修改成功";
        }
    }else if ( del != null && !del.equals("") ) {
        String delID = request.getParameter("bookid");
        if ( !cart.delShoper(delID,request) ) {
            msg = "删除清单中的图书时出错!" ;
        }
    }
    else if (clearCar != null && ! clearCar.equals("") ) {
        session.removeAttribute("cart");
        msg = "购物车中的物品清单已清空";
    }
%>
  <body>
<div align = "center">
    <table width = "750" border = "0" cellspacing = "1" cellpadding = "1">
    <tr>
        <td width = "200">  </td>
        <td width = "100"><a href = "booklist.jsp">在线购物</a></td>
        <td width = "100"><a href = "cartlist.jsp">我的购物车</a></td>
        <td><a href = "logout.jsp">用户注销</a></td>
```

```
    </tr>
    </table>

    <table width="900" border="0" cellspacing="1" cellpadding="1">
    <tr valign="top">
        <td align="center">
        <p><br>
        <b><font color="#0000FF"><% = (String)session.getAttribute("username") %>的购
物车物品清单</font></b></p>
<%
    if (!msg.equals("") )
        out.println("<p><font color=#ff0000>" + msg + "</font></p>");

        Vector cartlist = (Vector) session.getAttribute("cart");
        if (cartlist==null || cartlist.size()<0 ){
            if (msg.equals(""))
            out.println("<p><font color=#ff0000>你还没有选择购买图书!请先购买</font>
</p>");
        } else {
%>
      <table width="100%" border="1" cellspacing="1" cellpadding="1" >
        <tr align="center">
          <th width="180">图书名称</td>
          <th width="100">作者</th>
          <th width="160">书号</th>
          <th width="130">出版社</th>
          <th width="80">单价(元)</th>
          <th width="90">出版时间</th>
          <th width="60">数量</th>
          <th colspan =2 width="100">选择</th>
        </tr>
    <%
    float totalprice = 0;
    int totalamount = 0;
    for (int i=0; i<cartlist.size();i++){
        Item iList = (Item) cartlist.elementAt(i);
        if (bookutil.getOnebook((int)iList.getBookNo())!= null) {
            Vector new_book_list = bookutil.getOnebook((int)iList.getBookNo());
            Book book = (Book) new_book_list.elementAt(0);
            totalprice = totalprice + book.getPrice() * iList.getAmount();
            totalamount = totalamount + iList.getAmount();
    %>
        <tr>
            <td align="center"><% = book.getBookname() %></td>
            <td align="center"><% = book.getAuthor() %></td>
            <td align="center"><% = book.getIsbn() %></td>
            <td align="center"><% = book.getPublish() %></td>
            <td align="center"><% = book.getPrice() %></td>
            <td align="center"><% = book.getPubdate() %></td>
            <form name="change" method="post" action="booklist.jsp" onSubmit="return
check();">
```

```
            <td align = "center">
                <input type = "text" name = "amount" maxlength = "4" size = "3" value = "<% =
iList.getAmount() %>" >
            </td>
            <td align = "center" width = "55" >
              <input type = "hidden" name = "bookid" value = "<% = iList.getBookNo() %>" >
              <input type = "submit" name = "modi" value = "修改" >
            </td>
            </form>
            <form name = "del" method = "post" action = "cartlist.jsp" >
              <input type = "hidden" name = "bookid" value = "<% = iList.getBookNo() %>" >
              <td align = "center" width = "55">< input type = "submit" name = "del" value = "删
除"></td>
            </form>
          </tr>
        <% }
    } %><tr><td colspan = "9" align = "right"><br>你选择的图书的总金额:<% = totalprice %>元
  总数量: <% = totalamount %>本  </td></tr>
        </table>
      <p></p>
          <table width = "90 % " border = "0" cellspacing = "1" cellpadding = "1">
            <tr>
              <td align = "right" valign = "bottom"> <a href = "booklist.jsp">继续购书</a>
   
                <a href = "booklist.jsp?clear = 1">清空我的购物车</a>
              </td>
            </tr>
          </table>
<% } %>
        </td>
      </tr>
    </table>
</div>
    </body>
</html>
```

本章小结

本章介绍了如何使用 JSP 与 JavaBean 结合来开发应用程序，并通过一个用户信息查询的例子介绍了所涉及的知识点。JavaBean 的使用在本章是一个重点内容，通过学习可以看出，在使用 JSP 进行开发时，是离不开 JavaBean 支持的。本章介绍了两种使用 JavaBean 的方法，使用好 JavaBean 将对 JSP 开发起到至关重要的作用。

习题 7

一、简答题

1. 简述 JavaBean 的代码应遵循的规则。
2. 简述 JavaBean 的使用范围。

二、操作编程题

1. 创建一个数据库 mywork,在该数据库中创建好友信息表 friendaddress,包括好友姓名、性别、联系电话、E-mail、联系地址、QQ 等信息。编写好友录入功能模块、查询功能模块及删除好友功能模块。

2. 编写录入一个好友的 JSP 页面,使用 JavaBean Friend 来提取该页面的信息并显示出来。

第8章 Servlet编程技术

Servlet 是用 Java 编写的，运行在 Web 服务器上的独立模块。在实际应用中可以灵活地加载和卸下 Servlet 模块，以此提高 Web 服务器功能。本章主要讲解 Servlet 的特点、工作原理及 Servlet 的编程技术。

本章主要内容：

- Servlet 介绍；
- Servlet 运行环境；
- Servlet 与 JSP；
- 通过 Servlet 实现多层数据库应用程序。

8.1 Servlet 介绍

随着动态网页技术的日益发展，1996 年 Sun 公司推出了 Servlet。Java Servlet 的编程模式和 CGI 类似，但它的功能和性能要比 CGI 强大得多。Sun 公司 1999 年 6 月推出的 JSP 技术，就是基于 Java Servlet 以及整个 Java 体系的 Web 开发技术。

8.1.1 什么是 Servlet

Servlet 是用于 Web 服务器端的 Java 小程序，它在 Web 服务器端被解释执行，用于处理客户端的请求和产生动态网页内容。与传统的从命令行启动的 Java 应用程序不同，Servlet 由 Web 服务器进行加载，且该 Web 服务器必须包含支持 Servlet 的 Java 虚拟机。

Servlet 与协议和平台无关，运行于支持 Java 的 Web 服务器中，生成动态的 Web 页面，担当客户请求（Web 浏览器或其他 HTTP 客户程序）与服务器响应（HTTP 服务器上的数据库或应用程序）的中间层。

一个 Servlet 程序负责处理它所对应的一个或一组 URL 地址的访问请求，接收访问请求信息和产生响应内容。Servlet 程序具有如下一些基本功能：

（1）获取客户端通过 HTML 的 FORM 表单递交的数据和 URL 后面的参数信息。

（2）创建对客户端的响应消息内容。

（3）访问服务器端的文件系统。

（4）连接数据库并开发基于数据库的应用。

（5）调用其他的 Java 类。

与普通 Java 程序相比,Servlet 只是输入信息的来源和输出结果的目标不一样,因此普通 Java 程序所能完成的大多数任务,Servlet 程序都可以完成。

8.1.2 Servlet 的工作原理

Servlet 由支持 Servlet 的服务器引擎负责管理运行,该服务器引擎为每一个请求创建一个轻量级的线程并进行管理。Servlet 的工作原理如图 8-1 所示。

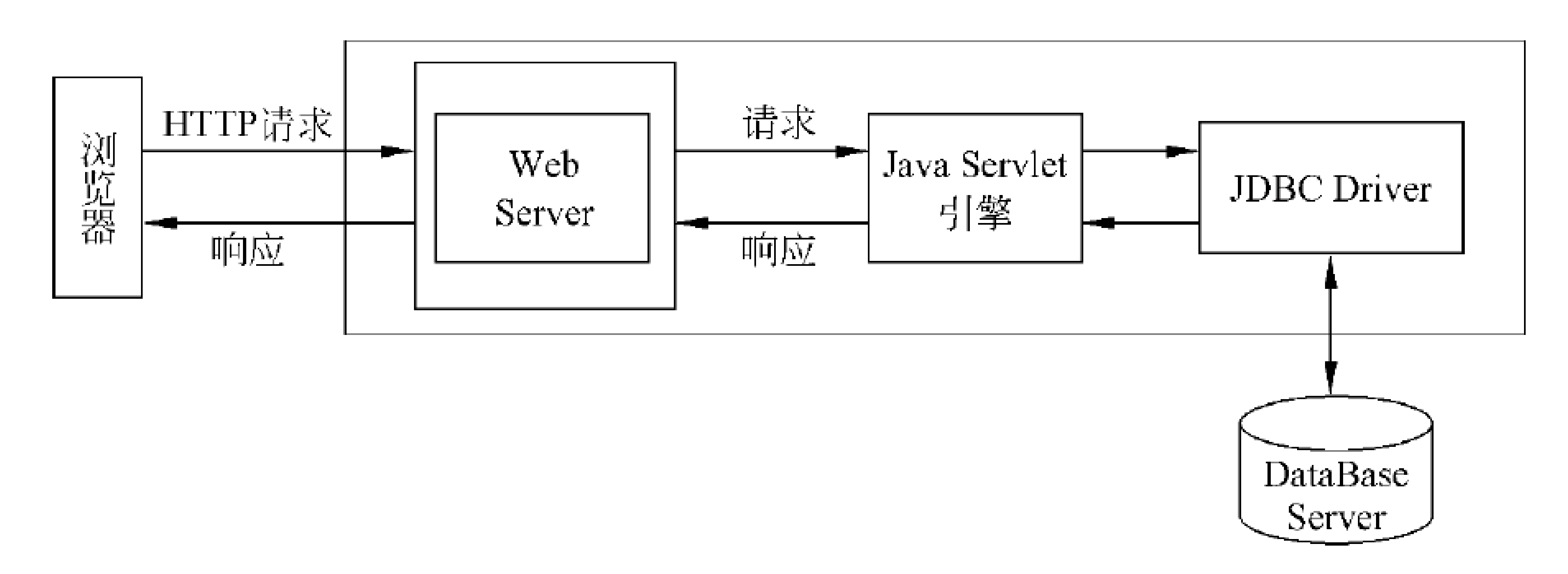


图 8-1　JSP 的工作原理

从图 8-1 中可以看出,整个处理流程如下:

(1) 浏览器向 Web 服务器发出请求。即使用浏览器按照 HTTP 协议输入一个 URL 地址,向 Web 服务器提出请求。

(2) Web 服务器响应该请求后,转交给 Servlet 引擎处理。

(3) Servlet 引擎检查对应的 Servlet 是否已装载,若没有装载,则将其载入内存并初始化,然后由该 Servlet 对请求进行处理。若 Servlet 中已含有访问数据库的操作,则还要通过相关的 JDBC 驱动程序与数据库相连,对数据库进行访问。

(4) Servlet 通过 JDBC 取回结果,生成 HTML 页面并将页面送回 Web 服务器。

(5) 最后 Servlet 将动态生成的标准 HTML 页面发送给客户端浏览器。

8.1.3 Servlet 的优点

Servlet 具备 Java 跨平台的优点,不受软硬件环境的限制,其具体优点如下:

1. 可移植性好

Servlet 是使用 Java 语言来编写的,因此,它延续了 Java 在跨平台上的表现,可以在不同的操作系统平台和不同应用服务器平台下移植。几乎所有的主流服务器都直接或通过插件间接支持 Servlet。

2. 高效

在传统的 CGI 中,客户机向服务器发出的每个请求都要生成一个新的进程。在 Servlet 中,每个请求将生成一个新的线程,而不是一个完整的进程。Servlet 被调用时,它被载入驻留在内存中,直到更改 Servlet,它才会被再次加载。

3. 功能强大

Servlet 可以使用 Java API 核心的所有功能,这些功能包括 Web 和 URL 访问、图像处

理、数据压缩、多线程、JDBC、RMI、序列化对象等。

4. 方便

Servlet 提供了大量的实用工具例程，例如，自动地解析和解码 HTML 表单数据、读取和设置 HTTP 头、处理 Cookie、跟踪会话状态等。

5. 可重用性

Servlet 提供重用机制，可以给应用建立组件或用面向对象的方法封装共享功能。

6. 模块化

JSP、Servlet、JavaBean 都提供把程序模块化的途径——把整个应用划分为许多离散的模块，各模块负责一项具体的任务，使程序便于理解。每一个 Servlet 可以执行一个特定的任务，Servlet 之间可以相互交流。

7. 节省投资

不仅有许多廉价甚至免费的 Web 服务器可供个人或小规模网站使用，而且对于现有的服务器，如果它不支持 Servlet，想要加上这部分功能也往往是免费的或只需要极少的投资。

8. 安全性

Servlet 可以充分利用 Java 的安全机制，并且可以实现类型的安全性。在 Java 的异常处理机制下，Servlet 能够安全地处理各种错误，不会因为程序上的逻辑错误而导致整体服务器系统的毁灭。

8.2 Servlet 程序的运行环境

运行一个 Servlet 程序，首先要将 Servlet 源文件编译为字节码文件，然后将字节码文件保存到相应的 Web 目录中，最后设置 Servlet 的调用路径，即配置 web. xml 文件。

8.2.1 编译 Servlet 程序

1. 创建用户目录

在编写 Servlet 类之前，首先要创建一个用户目录，用以保存 Servlet 源文件。本书创建一个目录为 E:\code\8。

2. 编写自己的 Servlet 类

用记事本工具编写一个简单的 Servlet 类，该类包含一个 init()方法和 service()方法，其功能是向客户端输出一个字符串。将该文件保存在 E:\code\8 目录下。程序 servletself. java 代码如下。

```
<% -- 例程 8-1 自己的 servlet 类 servletself.java -- %>

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class servletself extends HttpServlet
{
   public void init(ServletConfig config) throws ServletException
   {
      super.init(config);
   }
   public void service(HttpServletRequest reqest, HttpServletResponse response)
  throws IOException
   {
      response.setContentType("text/html;charset = GB2312"); //设置响应的类型
      PrintWriter out = response.getWriter();
      out.println("<HTML><BODY>");
      out.println("<font size = 6 color = red>将本字符串用红色字体输出到客户端</font>");
      out.println("</body></html>");
    }
}
```

【说明】

基于 HTTP 协议的 Servlet 必须导入 javax.servlet.* 和 javax.servlet.http.* 包。

(1) javax.servlet.*：存放与 HTTP 协议无关的一般性 Servlet 类。

(2) javax.servlet.http.*：除了继承 javax.servlet.* 之外，还增加了与 HTTP 协议有关的功能。

3. 获取 Servlet API 包

编译程序 Servletself.java 时，需要用到 Servlet API 基本包，这些包在文件 servlet-api.jar 中，在安装目录 Tomcat 6.0\lib 下找到该文件，并将该文件复制到 E:\code\8 目录下。

4. 编译 Servlet 源文件

在命令提示符窗口下，进入 E:\code\8 目录，编译程序 servletself.java。在 DOS 窗口中，输入命令：javac-classpath servlet-api.jar servletself.java。

编译后的字节码文件是 servletself.class。

8.2.2 存放 Servlet 字节码文件到相应目录

1. 部署 Servlet 字节码文件

Tomcat 6.0 服务器存放 Servlet 字节码文件的目录是 Tomcat 6.0\webapps\ROOT\WEB-INF\classes，将 servletself.class 文件复制到该目录下。

2. 配置 web.xml 文件

web.xml 文件在 ROOT\WEB-INF 目录下，编辑该文件。

在 http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"version="2.5 标记之后增加以下代码：

```
<servlet>
        <servlet-name>name_1</servlet-name>
        <servlet-class>servletself</servlet-class>
</servlet>

<servlet-mapping>
        <servlet-name>name_1</servlet-name>
        <url-pattern>/doget1</url-pattern>
</servlet-mapping>
```

8.2.3　运行 Servlet

保存 web.xml 文件后，重新启动 Tomcat 6.0 服务器，然后在浏览器地址栏中输入 http://localhost:8080/doget1 就可以访问该 Servlet 了，访问结果如图 8-2 所示。

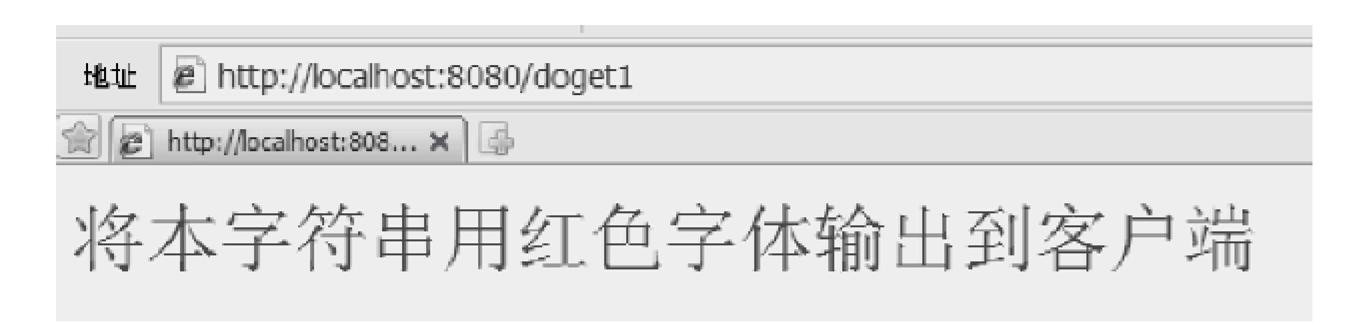


图 8-2　访问 Servlet 模块

8.3　Servlet 的基本结构

Servlet 模块是用 Servlet API 编写的。Servlet API 包含两个包，分别为 javax.servlet 和 javax.servlet.http。javax.servlet 包中的类与 HTTP 协议无关；javax.servlet.http 包中的类与 HTTP 协议相关，该包中的部分类继承了 javax.servlet 包中的部分类和接口。Servlet 的基本结构如图 8-3 所示。

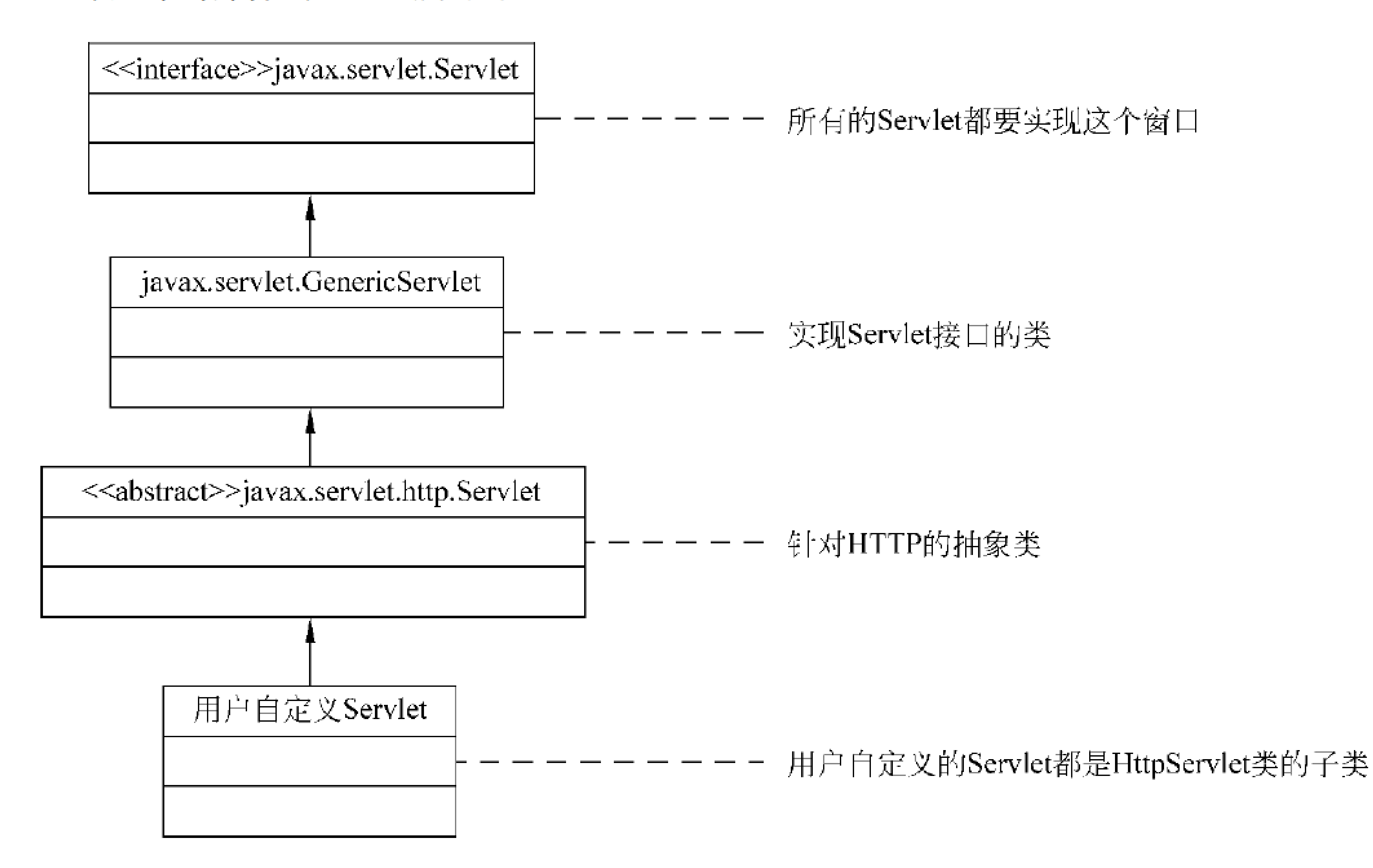


图 8-3　Servlet 的基本层次结构

8.3.1 Servlet 的成员方法

Servlet 程序必须实现 javax.servlet.Servlet 接口，Servlet 接口定义了 Servlet 容器与 Servlet 程序之间的通信协议。为了简化 Servlet 程序的编写，Servlet API 也提供了一个实现 Servlet 接口的 GenericServlet 类，这个类实现了 Servlet 程序的基本特征和功能。另外，Servlet API 中还提供了一个专用于 HTTP 协议的 HttpServlet 类。

1. GenericServlet 类

GenericServlet 类在 javax.servlet 包中，它提供了 servlet 接口的基本实现，该类包含 3 个重要的方法，分别是 init()方法、destroy()方法和 service()方法。其中，service()是抽象方法，所有子类都应当实现这个方法。

2. HttpServlet 类

HttpServlet 类是 GenericServlet 类的子类，在 GenericServlet 类的基础上进行了一些针对 HTTP 特点的扩充。因此，开发基于 Servlet 类的应用必须继承 GenericServlet 类或 HttpServlet 类。为了充分利用 HTTP 协议的功能，一般情况下，都将自己编写的 Servlet 作为 HttpServlet 类的子类。而 HttpServlet 类是一个抽象类，开发者必须在自己定义的继承类中实现 HttpServlet 类的所有方法。

HttpServlet 类定义了两个 service()方法和 6 个 doXXX()方法。

1） service()方法

service()方法是 Servlet 的核心，它是一个 Servlet 的 http-specific 方案，负责把请求分配给支持这个请求的其他方法。

第一个 service()方法为：

```
public void service ( ServletRequest request, ServletResponse response ) throws
ServletException, IOException;
```

本方法是公有方法。该方法接收客户端请求包后，创建 request 对象和 response 对象，并分别转换为 HttpServletRequest/HttpServletResponse 类型的对象，然后调用下面的保护 service()方法。

第二个 service()方法为：

```
protected void service (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException;
```

该方法接收 HttpServletRequest/HttpServletResponse 类型的对象后，根据 HTTP 请求方法的类型，本方法调用下面 6 个 doXXX()方法之一，进行逻辑处理并响应客户。

2） doXXX()方法

doXXX()方法有如下 6 个：

• doGet()方法

```
protected void doGet(HttpServletRequest request, HttpServletResponse response ) throws
ServletException,IOException;
```

该方法用来处理一个 HTTP GET 操作,这个操作允许客户端简单地从 HTTP 服务器“获得”资源。当一个客户通过 HTML 表单发出一个 HTTP GET 请求或直接请求一个 URL 时,doGet()方法被调用。与 GET 请求相关的参数添加到 URL 的后面,并与这个请求一起发送。

专家点拨: doGet()方法提交的数据量不能超过 2KB,否则提交失败。

- doPost()方法

```
protected void doPost (HttpServletRequest request, HttpServletResponse response) throws
ServletException,IOException;
```

该方法用来处理一个 HTTP POST 操作,这个操作包含请求体的数据。当一个客户通过 HTML 表单发出一个 HTTP POST 请求时,doPost()方法被调用,在开发者要处理 POST 操作时,必须在 HttpServlet 的子类中重载该方法。

- doDelete()方法

```
protected void doDelete(HttpServletRequest request, HttpServletResponse response) throws
ServletException,IOException;
```

该方法用来处理一个 HTTP DELETE 操作。这个操作允许客户端请求从服务器上删除 URL。当开发者要处理 DELETE 请求时,必须重载该方法。

- doOptions()方法

```
protected void doOptions(HttpServletRequest request,HttpServletResponse response) throws
ServletException,IOException;
```

该方法用来处理一个 HTTP OPTION 操作,这个操作自动地决定支持哪一种 HTTP 方法。

- doPut()方法

```
protected void doPut (HttpServletRequest request, HttpServletResponse response) throws
ServletException,IOException;
```

该方法用来处理一个 HTTP PUT 操作,这个操作类似于通过 FTP 发送文件,当要处理 PUT 操作时,必须在 HttpServlet 的子类中重载该方法。

- doTrace()方法

```
protected void doTrace(HttpServletRequest request, HttpServletResponse response) throws
ServletException,IOException;
```

该方法用来处理一个 HTTP TRACE 操作,该操作的默认执行结果是产生一个响应,这个响应包含一个反映 TRACE 请求中发送的所有头域的信息。当开发 Servlet 时,在多数情况下需要重载该方法。

每当客户请求一个 HttpServlet 对象时,该对象的 service()方法就会被调用,而且传递给该方法和一个“请求”(ServletRequest)对象以及一个“响应”(ServletResponse)对象作为参数,在 HttpServlet 中已存在。默认的服务功能是调用与 HTTP 请求的方法相应的 doXXX()方法,如 HTTP 请求的方法为 get,则默认情况下就调用 doGet()。

程序员自定义的 Servlet 类都必须扩展 HttpServlet 类。在扩展类中,覆盖 service()、doPost()或 doGet()方法来实现逻辑处理。

【例 8-1】 获取客户端信息,并把信息输出到客户端。

本例中的 guestinif.jsp 页面提供一个信息输入窗口,并把信息提交给 Servlet(guestinif.java)处理,程序源代码如下。

```
<! -- 例程 8 - 2 guestinif.jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<HTML>
<BODY>
  <FORM ACTION = "/doget2" method = "post">
     请输入姓名:
     <INPUT TYPE = "TEXT" NAME = "myname"><br>
     您的兴趣:
    <SELECT NAME = "love">
       <OPTION value = "Sleep">Sleep </OPTION>
       <OPTION value = "Dance">Dance </OPTION>
       <OPTION value = "Travel">Travel </OPTION>
    </SELECT><br>
    <INPUT TYPE = "SUBMIT" NAME = "mysubmit" value = 提交><br>
    <INPUT TYPE = "RESET" VALUE = "重新填写"><br>
  </FORM>
</BODY>
</HTML>
```

```
<! -- 例程 8 - 3 guestinif.java -->

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class guestinif extends HttpServlet
{
  //重写 doPost 方法
  public void doPost(HttpServletRequest req, HttpServletResponse res)
  throws ServletException,IOException
  {
      res.setContentType("text/html;charset = GB2312");     //设置输出流的文件类型
      PrintWriter out = res.getWriter();                     //获得输出流对象
      req.setCharacterEncoding("GB2312");
      String name = (String)req.getParameter("myname") ;
      String love = (String)req.getParameter("love") ;
      out.println("<html><body>");
      out.println("名字:" + name);                            //获得客户端"myname"的输入信息
      out.println("<br>");
      out.println("爱好:" + love);                            //获得客户端"love"的输入信息
      out.println("</body></html>");
      out.close();
  }
}
```

程序运行后的效果如图 8-4 所示。其中左图为 guestinif.jsp 的运行效果,输入姓名和兴趣并单击“提交”按钮后,显示右图的内容。本例源代码存放于本书配套素材中的 guestinif.jsp 及



图 8-4 信息输出的效果

guestinif. java 文件中(文件路径为 code\8)。

8.3.2 Servlet 的生命周期

当服务器调用 Servlet 类时,Servlet 对象被创建。从服务器创建 Servlet 对象到该对象被消灭这段时间称为 Servlet 生命周期。

Servlet 的生命周期分为装载 Servlet、处理客户请求和结束 Servlet 3 个阶段,分别由 javax. servlet. Servlet 接口的 init()方法、service()方法和 destroy()方法来实现。

1. 装载 Servlet

所谓装载 Servlet,实际上是用 Web 服务器创建一个 Servlet 对象,调用这个对象的 init()方法完成必要的初始化工作。在 Servlet 对象生命周期内,本方法只调用一次。

2. 处理客户请求

当客户请求到来时,Servlet 引擎将请求对象传递给 service()方法,同时创建一个响应对象,service()方法获得请求/响应对象后,进行请求处理(调用被覆盖的 doXXX()方法进行逻辑处理),然后将处理的结果以响应对象的方式返回给客户端。在 Servlet 对象周期内,该方法可能被多次请求,而被多次调用。

Servlet 的响应有以下类型:

(1) 一个输出流,浏览器根据它的内容类型(如 TEXT/HTML)进行解释。

(2) 一个 HTTP 错误响应,重定向到另一个 URL、Servlet、JSP。

3. 结束 Servlet

当 Web 服务器要卸载 Servlet 或重新装入 Servlet 时,服务器会调用 Servlet 的 destroy()方法,将 Servlet 从内存中删除,否则它一直为客户服务。在 Servlet 对象周期内,该方法只调用一次。

Servlet 的生命周期时序图如图 8-5 所示。开始于将它装载到 Web 服务器,结束于终止或重新装载 Servlet。当 Servlet 被加载后,主要通过循环调用 service()方法为用户服务。

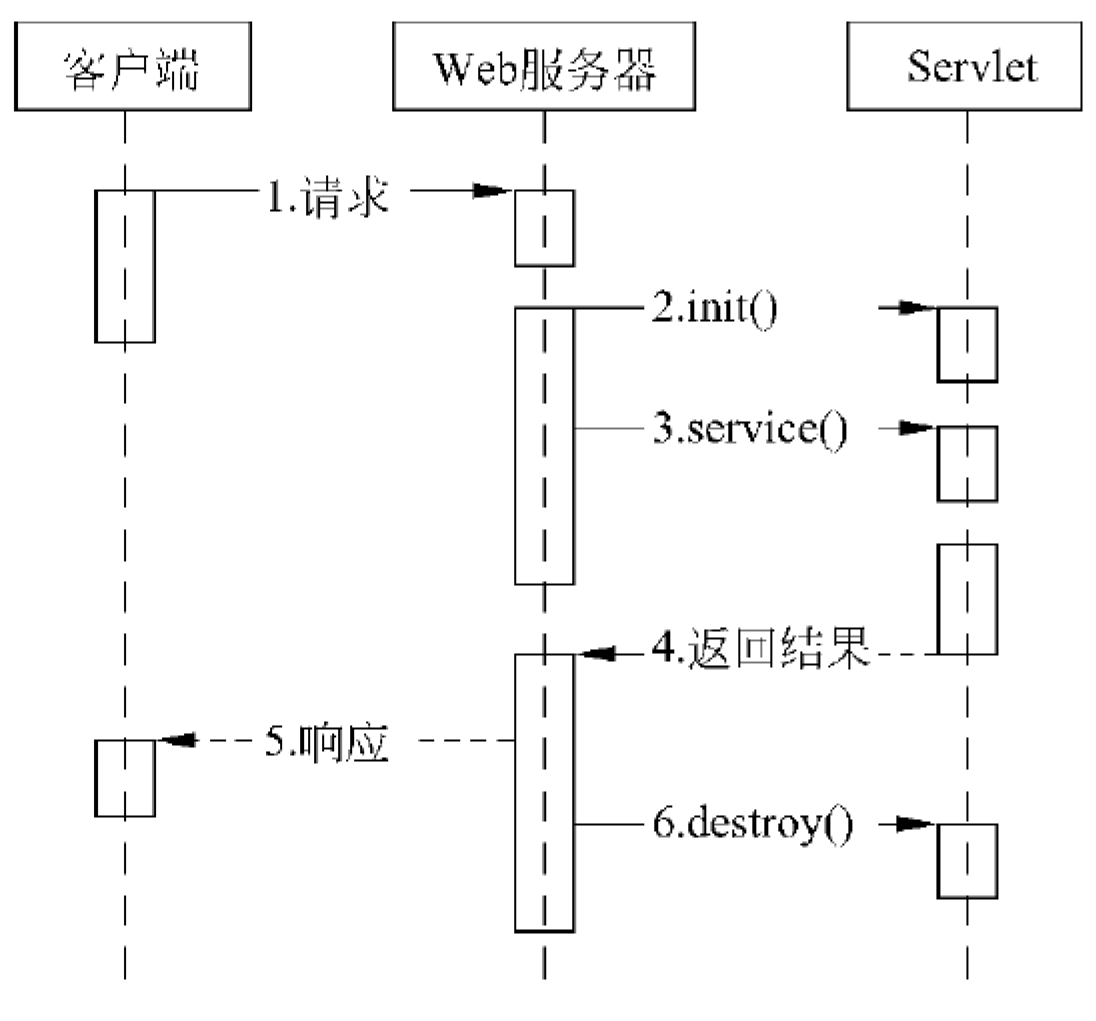


图 8-5 Servlet 生命周期

【例 8-2】 查询学生成绩。

本例由一个 JSP(stu.jsp)页面和一个 Servlet(stu.java)模块组成，以按姓名查询学生(表：students)成绩为例，说明 Servlet 对象的 3 个基本方法的作用。

在 JSP 页面中，由表单提供姓名录入窗口，将姓名提交给 Servlet 模块，该 Servlet 模块查询出学生成绩，并把成绩输出到客户端，程序源代码如下。本例源代码存放于本书配套素材中的 stu.jsp 及 stu.java 文件中(文件路径为 code\8)。

```
<! -- 例程 8 - 4 stu.jsp -->

<%@ page contentType = "text/html;charset = GB2312" %>
<HTML>
<BODY>
<FONT size = 3>
   <P><b>成绩查询</b><br>
   <FORM action = "/doget4" Method = "post">
     <P>输入姓名:
     <Input type = text name = "name">
     <Input type = submit name = "g" value = "提交">
   </Form>
</FONT>
</BODY>
</HTML>
```

```
<! -- 例程 8 - 5 stu.java -->

import java.io. * ;
import javax.servlet. * ;
import javax.servlet.http. * ;
import java.sql. * ;
public class stu extends HttpServlet
{
   Connection con = null; //声明一个连接对象
   public void init(ServletConfig config) throws ServletException
   {
      super.init(config);
      try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");//加载驱动程序
        }
      catch(ClassNotFoundException e)
        { }
      try{
            con = DriverManager.getConnection("jdbc:odbc:grade");//创建连接对象
        }
      catch(SQLException e)
        { }
   }

//重写 service 方法
   public void service(HttpServletRequest req, HttpServletResponse res)
```

```
throws ServletException,IOException
{
    res.setContentType("text/html;charset = GB2312"); //设置输出流的文件类型
    PrintWriter out = res.getWriter(); //获得输出流对象
    req.setCharacterEncoding("GB2312");
    String name = req.getParameter("name"); //获取提交的姓名
    name = name.trim();
    if(name == null)
      {
        name = "";
      }
    Statement sql = null;
    ResultSet rs = null;
    try
      {
          sql = con.createStatement();
          String condition = "SELECT * FROM students WHERE name = " + "'" + name + "'";
          rs = sql.executeQuery(condition);
          out.println("<html><body bgcolor = 'pink'>");
          out.println("<center>");
          out.println("查询结果如下");
          out.print("<Table Border>");
          out.print("<TR>");
          out.print("<TH width = 100>" + "学号");
          out.print("<TH width = 100>" + "姓名");
          out.print("<TH width = 50>" + "数学成绩");
          out.print("<TH width = 50>" + "英语成绩");
          out.print("<TH width = 50>" + "物理成绩");
          out.print("</TR>");
          while(rs.next())
            {
              out.print("<TR>");
              out.print("<TD>" + rs.getString(1) + "</TD>");
              out.print("<TD>" + rs.getString(2) + "</TD>");
              out.print("<TD>" + rs.getInt(3) + "</TD>");
              out.print("<TD>" + rs.getInt(4) + "</TD>");
              out.print("<TD>" + rs.getInt(5) + "</TD>");
              out.print("</TR>");
            }
          out.print("</Table>");
          out.println("</center>");
          out.println("</body></html>");
          out.close();
      }
  catch(SQLException e)
      {}
}
public void destroy()
{
    super.destroy();
    try
```

```
            {
                con.close();
            }
          catch(Exception e)
            {
              e.printStackTrace();
            }
        }
    }
```

【说明】

(1) init()方法作用：由于存在多个客户与同一数据库连接的事实。因此 Servlet 对象初始化要做的工作是加载驱动程序，创建连接对象 con。

(2) service()方法的作用：由于存在多个客户，多次查询学生成绩的事实(请求)，因此将查询学生成绩的实现代码放置在 service()方法中。该方法以姓名为关键字，查询学生成绩，并将成绩输出到客户端。

(3) destroy()方法的作用：当客户希望不再查询时，调用 destroy()方法释放资源，销毁 Servlet 对象。

本例源代码存放于本书配套素材中的 stu.jsp 及 stu.java 文件中(文件路径为 code\8)。

8.4 Servlet 与 JSP

JSP 是继 Servlet 后 Sun 公司推出的新技术，它是以 Servlet 为基础开发的，JSP 经过转译后的结果就是 Servlet。通常，JSP 适合于展示用户界面，Servlet 适合于企业逻辑处理。因此，开发者常使用 JSP 技术编写用户界面，使用 Servlet 实现业务逻辑。

Servlet 与 JSP 区别如下所述。

(1) 编程方式不同：JSP 是为了解决 Servlet 中相对困难的编程技术而开发的技术，因此，JSP 在程序的编写方面比 Servlet 要容易得多，Servlet 严格遵循 Java 语言的编程标准，而 JSP 则遵循脚本语言的编制标准。

(2) Servlet 必须在编译以后才能执行：JSP 并不需要另外进行编译，JSP Container 会自动完成这一工作，而 Servlet 在每次修改代码之后都需要编译完才能执行。

(3) 运行速度不同：由于 JSP Container 将 JSP 程序编译成 Servlet 时需要一些时间，所以 JSP 的运行速度比 Servlet 要慢一些，不过，如果 JSP 文件能毫无变化地重复使用，它在第一次以后的调用中运行速度就会和 Servlet 一样了，这是因为 JSP Container 接到请求以后会确认传递过来的 JSP 是否有改动，如果没有改动，将直接调用 JSP 编译过的 Servlet 类，并提供给客户端解释执行，如果 JSP 文件有所改变，JSP Container 将重新将它编译成 Servlet，然后提交给客户端。

8.4.1 在 Servlet 和 JSP 页面共享信息

要在 Servlet 和 JSP 页面之间共享信息，可以通过以下 5 种方法完成。

(1) 通过 Session(HttpSession)把数据信息保存在 Session 中。

（2）通过 ServletContext 对象。

（3）通过 Application 对象。

（4）通过隐含的表单把数据信息提交到下一个页面。

（5）通过文件系统或数据库。

本节主要介绍通过 ServletContext 对象共享信息的方法。

ServletContext 是定义在 Javax. servlet 包中的对象。它定义了用于 Web 应用中的服务器端组件关联 Servlet 容器的方法集合。ServletContext 经常被用于存储对象的区域，这些对象在 Web 应用中的所有的服务器端组件中使用。可以把 ServletContext 当作在 Web 应用中共享的存储区域。把一个对象放置到 ServletContext 中时，它存在于 Web 应用的整个生命周期中，除非它被明确地删除或替换。在 ServletContext 中定义了 4 个方法来实现存储区的共享功能。

- setAttribute(String name，Object obj)：通过名称绑定一个对象并存储对象到当前 ServletContext。如果指定的名称已经被使用过，这个方法会删除旧对象而绑定新对象。
- getAttribute(String name)：返回指定名称的对象，如果名称不存在，则返回 NULL。
- removeAttribute(String name)：从 ServletContext 中删除指定名称的对象。
- getAttributeNames()：返回在 ServletContext 中指定名称的对象集合。

在 JSP 页面中可以通过 getServletContext()方法来获得 ServletContext 对象。下面就通过 ServletContext 创建的一个简易聊天室，来介绍 Servlet 和 JSP 页面之间是如何共享信息的。

【例 8-3】 简易聊天室 liaotian. jsp 文件源代码如下。

```
<%  -- 例程 8 - 6 liaotian.jsp --  %>

<%@ page language = "java" import = "java.util.*,javax.servlet.*,
javax.servlet.http.*" pageEncoding = "gb2312" %>
<%
String path  =  request.getContextPath();
String basePath  =  request.getScheme() + "://" + request.getServerName() + ":" +
request.getServerPort() + path + "/";
request.setCharacterEncoding("gb2312");
 %>
<!DOCTYPE HTML PUBLIC " - //W3C//DTD HTML 4.01 Transitional//EN">
<html>
   <head>
     <base href = "<% = basePath %>">
     <title>聊天室</title>
     <meta http-equiv = "pragma" content = "no-cache">
     <meta http-equiv = "cache-control" content = "no-cache">
     <meta http-equiv = "expires" content = "0">
     <meta http-equiv = "keywords" content = "keyword1,keyword2,keyword3">
     <meta http-equiv = "description" content = "This is my page">
    </head>
    <body>
```

```
<h1 align="center">聊天室</h1>
<br>
<hr>
<font color="blue">
<%
        String content=(String)getServletContext().getAttribute(new String("content"));
        out.println(content);
getServletContext().setAttribute("content",content+request.getParameter("content")+"<br>");
 %>
</font>
<hr>
<form action="liaotian.jsp">
<input type="text" name="content">
<input type="submit" value="发言"><input type="reset" value="重置">
</form>
  </body>
</html>
```

在该程序段中第三对<% %>代码实现了聊天的方法。当单击"发言"按钮提交聊天内容后，将获得 ServletContext 中的 content 属性的值，即当前聊天的内容，然后把这个值显示在浏览器中，最后再将当前用户聊天的内容一起设置为 ServletContext 中的 content 属性的值。该程序运行结果如图 8-6 所示。本例源代码存放于本书配套素材中的 liaotian.jsp 文件中(文件路径为 code\8)。



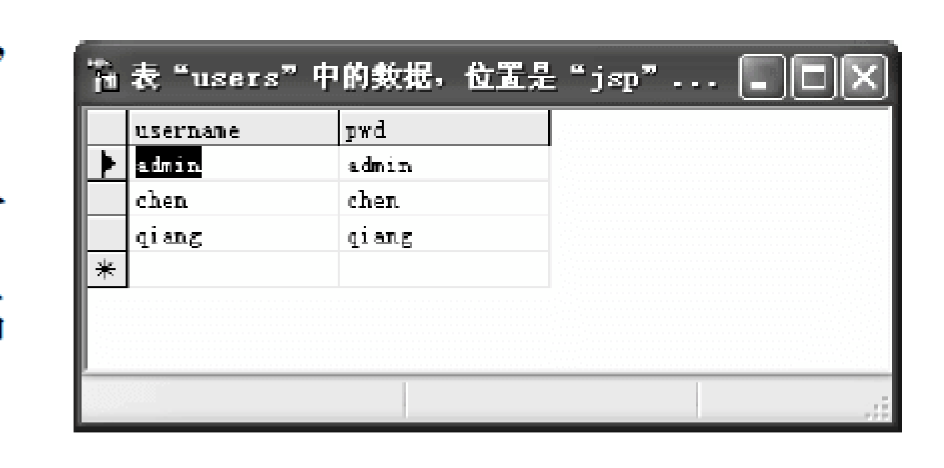
图 8-6　聊天室运行效果

8.4.2　在 JSP 中通过 Servlet 访问数据库

要在 Web 组件之间实现信息共享，并且在对一个 Web 应用被调用的间隙内维持数据通常是由一个数据库来维护的。Web 组件使用 JDBC API 来访问关系数据库。

【例 8-4】 在 Servlet 中用 JDBC 查询一个数据库，将结果返回到客户端并显示出来。

该程序使用了 JDBC 驱动程序，数据库采用了 SQL Server 2000 数据库，数据库名为 jsp，数据库表名为 users，如图 8-7 所示。



username	pwd
admin	admin
chen	chen
qiang	qiang

图 8-7　数据库表 users 的内容

本例首先要选取数据库的 Connection 对象，并创

建 Statement 对象，然后创建表示查询的命令字符串，再调用 executeQuery()方法，最后取得返回的 ResultSet 对象，并将取得的值输出。该程序访问数据库的 Servlet 文件 ConnectDb.java 源代码如下。

<! -- 例程 8-7 ConnectDb.java -->

```
package servlet;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.sql.*;
public class ConnectDB extends HttpServlet
 {
    Connection con = null;
    int flag = 0;
    public ConnectDB() {
        super();
    }
    public void destroy() {
        super.destroy();
        //关闭数据库连接
        try{
            con.close();
        }catch(SQLException e){
            e.printStackTrace();
        }
    }
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset = gb2312");
        PrintWriter out  =  response.getWriter();
        switch (flag){
            case 0:{
             try{
              out.println("<html>");
              out.println("<head><title>通过 servlet 数据库访问</title></head>");
              out.println("<body>");
              out.println("数据库连接成功!");
              Statement stmt = null;
              ResultSet rs = null;
                String sql = "select  *  from users";
              stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_
              READ_ONLY);
              rs = stmt.executeQuery(sql);
              int count = 0;
              out.println("<table border = '1'>");
              out.println("<tr>");
```

```
                out.println("<td bgcolor = '#cccfff'>用户名</td>");
                out.println("<td bgcolor = '#cccfff'>密码</td>");
                out.println("</tr>");

               //显示数据库表中的信息
              while (rs.next()){
                  out.println("<tr>");
                  out.println("<td bgcolor = '#ffcc33'>");
                  String username = rs.getString("username");
                  out.println(username);
                  out.println("</td>");
                  out.println("<td bgcolor = '#ffcc33'>");
                  String pwd = rs.getString("pwd");
                  out.println(pwd);
                  out.println("</td>");
                  out.println("</tr>");
                  count++;
              }
                  out.println("</table>");
                  out.println("<br>共查询记录: " + count + "个");
                  out.println("<p>目标数据库为: </p>");
                  out.println("</body>");
                  out.println("</html>");

              //关闭 ResultSet 对象
                  rs.close();
                  stmt.close();
                  break;
                  }catch(Exception e){
                      e.printStackTrace();
                  }
              }
              case 1:{
                  out.println("<html><body");
                  out.println("<p>不能加载 JDBC 驱动程序!</p>");
                  out.println("</body></html>");break;
              }
              case 2:{
                  out.println("<html><body>");
                  out.println("<p>不能创建数据库的连接!</p>");
                  out.println("</body></html>");
              }
          }
          out.flush();
          out.close();
      }
      public void doPost(HttpServletRequest request, HttpServletResponse response)
              throws ServletException, IOException {
      }
      public void init() throws ServletException {
          String drivername = "com.microsoft.jdbc.sqlserver.SQLServerDriver";
```

```
        String url = "jdbc:microsoft:sqlserver://localhost:1433;DatabaseName = jsp";
        String username = "sa";
        String password = "";
        try{
            //加载 JDBC 驱动程序
            Class.forName(drivername);
            //连接数据库
            con = DriverManager.getConnection(url,username,password);
        }catch(ClassNotFoundException e){
            flag = 1;
        }catch(SQLException e){
            flag = 2;
        }
    }
}
```

此例运行效果如图 8-8 所示。本例源代码存放于本书配套素材中的 ConnectDb.java 文件中(文件路径为 code\8)。

数据库连接成功!

共查询记录:
3个

目标数据库为:

用户名	密码
admin	admin
chen	chen
qiang	qiang

图 8-8 Servlet 访问数据库运行效果

8.4.3 JSP 调用 Servlet

JSP 和 Servlet 是两种极具特色的动态 Web 技术,如果撇开底层运行机制上的共同之处(JSP 被翻译成 Servlet 再执行),单从开发人员的角度来看,完全可以单独采用其中一种技术实现一个动态 Web 应用。由于 Servlet 输出 HTML 时采用 CGI 的方式,是一句一句输出的,所以,编写和修改 HTML 文件非常不方便。而 JSP 把 Java 代码嵌套到 HTML 语句中,大大简化和方便了网页的设计和维护。因此,在开发 Web 应用实践中,主要是整合这两种技术,实现两种技术的优势互补。

在整合技术中,表示层的工作由 JSP 技术承担,注重页面的表现,编写输出 HTML 网页的程序;业务逻辑层的工作由 Servlet 承担,注重业务逻辑的实现,编写完成诸如数据计算、数据分析、数据库连接等操作处理的程序。

在 JSP 文件中访问 Servlet 所采用的格式与 HTML 页面中调用 Servlet 的方法完全一样,而且原理也完全相同。只不过这时访问 Servlet 的是动态的 JSP 文件,而不是静态的 HTML 页面。

显然,在 JSP 文件中访问 Servlet 也需要编写两个程序,即一个 JSP 程序和一个 Servlet 程序。下面通过一个 JSP+Servlet 程序来进行讲解。

【例 8-5】 编写一个调用 Servlet 的登录页面程序。

本例中的 login.jsp 程序是一个 Web 登录页面,在文本框中输入用户名和密码,单击“登录”按钮后,则把输入的信息提交给<FORM>标记中调用的 Servlet。程序 LoginServlet.java 是<FORM>标记中调用的 Servlet,它接收用户输入的信息并将处理结果输出到用户界面上。

登录页面 login.jsp 的源代码如下。

```
<! -- 例程 8 - 8 login.jsp -- >

<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<%
String path = request.getContextPath();
String basePath = request.getScheme() + "://" + request.getServerName() + ":"
 + request.getServerPort() + path + "/";
 %>

<!DOCTYPE HTML PUBLIC " - //W3C//DTD HTML 4.01 Transitional//EN">
<html>
   <head>
     <base href = "<% = basePath %>">
     <title>在 JSP 中调用 Servlet</title>
     <meta http - equiv = "pragma" content = "no - cache">
     <meta http - equiv = "cache - control" content = "no - cache">
     <meta http - equiv = "expires" content = "0">
     <meta http - equiv = "keywords" content = "keyword1,keyword2,keyword3">
     <meta http - equiv = "description" content = "This is my page">
     <! --
     <link rel = "stylesheet" type = "text/css" href = "styles.css">
      -->
   </head>
   <body>
     <form method = "post" action = "/code/8/servlet/LoginServlet" >
     <table align = "center" width = "240" cellspacing = "1" cellpadding = "1" border = "0"
bgColor = "#c4cab5">
         <tr>
             <td colspan = "2" align = "center">用户登录</td>
         </tr>
         <tr>
             <td>用户名: </td>
             <td><input type = "name" name = "username" size = "20"></td>
         </tr>
         <tr>
             <td>密码: </td>
             <td><input type = "password" name = "pwd" size = "20"></td>
         </tr>
         <tr>
             <td colspan = "2" align = "center"><input type = "submit" VALUE = "确定">  
 <input type = "reset" value = "重置"></td>
         </tr>
      </table>
        </form>
    </body>
</html>
```

【说明】

```
<form method = "post" action = "/code/8/servlet/LoginServlet ">
```

由于在 Web. xml 中我们将 LoginServlet 的 URL 形式设置为/LoginServlet，在 login. jsp 中指定将表单提交给/code/8/servlet/LoginServlet，因此用户单击“提交”按钮时，该请求将提交给 LoginServlet。

LoginServlet. java 的功能是测试用户名和登录密码，将处理结果输出到页面上，其源代码如下。

<! -- 例程 8 - 9 LoginServlet. java -->

```
package servlet;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class LoginServlet extends HttpServlet {
    public LoginServlet() {
        super();
    }
    public void destroy() {
        super.destroy();
    }
    public void doGet (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doPost(request,response);
    }
    public void doPost (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html;charset = GB2312");
        PrintWriter out = response.getWriter();
        byte b1[] = request.getParameter("username").getBytes("ISO - 8859 - 1");
        String username = new String(b1);
        String pwd = request.getParameter("pwd");
        out.println("<html><head></head><body>");
        out.println("<h3>输出客户端的信息</h3><br>");
        if(username.equals("chen") && pwd.equals("123456")) {
            out.println("用户名: " + username + "<br>");
            out.println("登录成功!<br>");
        }
        else
        {
            out.print("<p align = 'center'>");
            out.println("<a href = " + "login.jsp" + ">用户名或密码错,请重新输入!</a></p>");
        }
        out.println("</body></html>");
        out.flush();
        out.close();
    }
    public void init() throws ServletException {
        }
}
```

【说明】

(1) String pwd=request.getParameter("pwd");

在 LoginServlet 中,利用 HttpServletRequest 类的 getParameter()方法来取得由网页 login.jsp 文本域框 input 传来的数据。上述代码是用于获得网页提交的密码数据。

(2) PrintWriter out = response.getWriter();

```
byte b1[] = request.getParameter("username").getBytes("ISO-8859-1");
```

数据通过 HTTP 协议传输时会被转码,因此在接收时,必须再做转码的工作,才能够正确地接收到数据,否则得到的数据将是乱码,上述代码 getBytes("ISO-8859-1")就是将接收到网页的用户名(username)数据转换成中文模式。

该例程序的运行效果如图 8-9 所示。当输入的用户名和密码正确时,显示如图 8-10 所示的效果。若输入的用户名和密码不正确时,则显示如图 8-11 所示的效果。本例源代码存放于本书配套素材中的 login.jsp 及 Login Servlet.java 文件中(文件路径为 code\8)。



图 8-9 login.jsp 运行效果

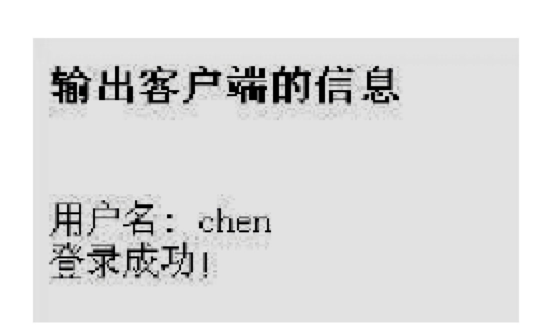


图 8-10 LoginServlet 运行效果

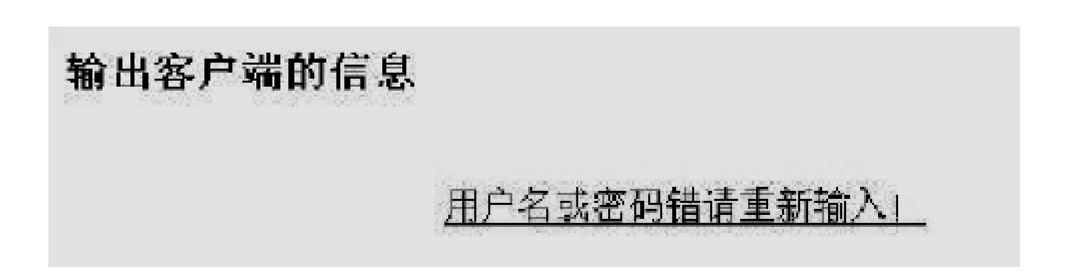


图 8-11 LoginServlet 运行登录错误效果

8.5 通过 Servlet 实现多层数据库应用程序

随着 Web 技术的深入和发展,传统的 C/S(客户机/服务器)两层结构的企业级应用系统已逐渐为 B/S(表示层/应用层/数据层)多层结构所取代,这种结构改变无论在 Microsoft 的 Windows DNA 中还是在以 Java 技术为核心的应用中都得到了具体的体现。Servlet 技术的出现推动了以 Java 为核心技术的多层 Web 应用的发展。

8.5.1 B/S 多层结构

B/S 结构(Browser/Server 结构)即浏览器和服务器结构,如图 8-12 所示。它是随着 Internet 技术的兴起,对 C/S 结构的一种变化或者改进的结构。在这种结构下,用户工作界面是通过 WWW 浏览器来实现的,极少部分事务逻辑在前端(Browser)实现,但是主要事务逻辑在服务器端(Server)实现,形成所谓三层结构。这样就大大简化了客户端电脑负荷,减轻了系统维护与升级

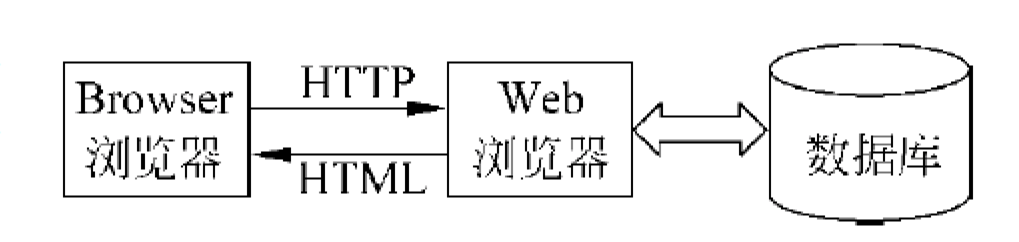


图 8-12 B/S 三层结构

的成本和工作量,降低了用户的总体成本。以目前的技术看,局域网建立 B/S 结构的网络应用,并通过 Internet/Intranet 模式下数据库应用,相对易于把握、成本也是较低的。它是一次性到位的开发,能实现不同的人员,从不同的地点,以不同的接入方式(如 LAN、WAN、Internet/Intranet 等)访问和操作共同的数据库;它能有效地保护数据平台和管理访问权限,服务器数据库也很安全。

8.5.2 数据层

数据库服务器在数据层中包含系统的数据处理逻辑,位于数据库服务器端。它的任务是接受 Web 服务器对数据库操作的请求,实现对数据库查询、修改、更新等功能,把运行结果提交给 Web 服务器。应用层可以通过 JDBC 来访问数据层。

8.5.3 应用层

具有应用程序扩展功能的 Web 服务器在功能层中包含系统的事务处理逻辑,运行在服务器中,是联系表示层与数据层的纽带。它的任务是接受用户的请求,首先需要执行相应的扩展应用程序与数据库进行连接,通过 SQL 等方式向数据库服务器提出数据处理申请,而后数据库服务器将数据处理的结果提交给 Web 服务器,再由 Web 服务器传送回客户端。这一层通常以动态链接库的形式存在并注册到服务器的 Registry 中,它与客户端通信的接口符合某一特定的组件标准(如 COM)。目前可用于实施应用层的技术 CGI、Java 及 Servlet 等。由于 Servlet 由 Web 服务器进行加载,利用 Java 语言进行开发,它在性能、可靠性以及可移植性等方面均比 CGI 有了长足的进步,因此 Servlet 是目前最适合实现应用层的技术。

8.5.4 表示层

Web 浏览器是三层结构中的第一个层次,在表示层中包含系统的显示逻辑,位于客户端。利用 Web 浏览器作为客户端,使客户端对应一个统一的应用界面。它的任务是由 Web 浏览器向网络上的某一个 Web 服务器提出服务请求,Web 服务器对用户身份进行验证后用 HTTP 协议把所需的主页传送给客户端,客户机接受传来的主页文件,并把它显示在 Web 浏览器上。

该层负责处理用户的输入和输出,但并不负责解释其含义。它可以利用 Windows 操作系统提供的 API 和应用服务器提供的各种服务建立丰富的图形用户界面,它们可以是用 VB、VC、Delphi 等编写的图形用户界面 GUI,也可以是 ASP、DHTML 等。它们主要提供给用户查询、维护数据的界面等。

8.5.5 多层应用程序的优点

B/S 多层结构从根本上改变了 C/S 结构的缺陷,是应用系统体系结构深刻的变革,它具有如下突出优点。

(1) 客户端不再负责数据库的存取和复杂数据计算等任务,只需要对其进行显示,充分发挥了服务器的强大作用,这样就大大地降低了对客户端的要求,降低了投资和使用成本。

(2) 易于维护、易于升级。维护人员不再为程序的维护工作奔波于每个客户机之间，而把主要精力放在功能服务器上。由于用户端无须专用的软件，当企业对网络应用进行升级时，只需更新服务器端的软件，从而减轻了系统维护与升级的成本与工作量。在这里体现了面向对象编程的特性之一——封装性的特点，而这一点在开发大型应用程序时尤其有用。可以把开发人员分成两组：一组负责开发系统界面层；另一组负责开发商业数据逻辑层。双方只要按照事先约定好的函数接口并行地开发就可以，而不必像以前那样，后面的工作必须等前面的工作完成后才能开始。

(3) 用户操作使用简便。B/S 结构的客户端只是一个提供友好界面的浏览器，通过鼠标即可访问文本、图像、声音、电影及数据库等信息，用户无须培训便可直接使用，利于推广。

(4) 更适合于网上发布信息。B/S 结构使用的是 Internet 的 Web 技术，因而更适合网上信息的发布，拓展了传统的数据库应用的功能，更适合 Internet 时代的需要。

然而，B/S 结构相对 C/S 结构也有其弱点，主要表现在：由于是三层的结构，网络流量不仅包括客户机与 Web 服务器之间的流量，而且包括 Web 服务器与数据库服务器之间的流量，因而网络流量较大，运行速度较慢。

8.6 上机指导

8.6.1 JSP 调用 Servlet 应用实例

1. 练习目标

(1) 熟悉 Servlet 类的 do×××()方法。
(2) 熟悉 Servlet 的生命周期。

2. 练习指导

本程序由一个 JSP 页面程序 diaoyong.jsp 构造一个表单接受客户输入的数据，提供姓名、性别、电话录入窗口，当提交表单后，把姓名、性别、电话数据提交给 Servlet 模块 diaoyong.java，Servlet 模块接受数据后，把数据写入到文件 phone.txt 中。

1) diaoyong.jsp

创建一个表单，包含 3 个文本框，分别用来输入姓名(name)、性别(sex)、电话(telphone)。该程序的源代码如下。

<!-- 例程 8-10 diaoyong.jsp -->

```
<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY bgcolor=cyan><FONT size=3>
  <FORM action="/doget5" method="post" name=form>
      姓名:<INPUT type="text" name="name"><br>
       性别:<INPUT type="text" name="sex"><br>
       电话:<INPUT type="text" name="telphone"><br>
       <input type=submit value=提交>
```

```
  </FORM>
</FONT>
</BODY>
</HTML>
```

2）diaoyong.java

（1）扩展类 HttpServlet，构造类 diaoyong。

（2）定义两个成员变量：ou（字符文件输出流），outbuff（字符缓冲输出流）。

（3）在 init()方法中，创建字符文件输出流和字符缓冲输出流，分别初始化成员变量 ou 和 outbuff。

（4）在 doPost()方法中，设置向客户端输出类型为 text/html;charset=GB2312。

（5）获取向客户端的输出流为 out。

（6）设置字符编码为 GB2312。

（7）获取表单中的数据：name/sex/telphone。

（8）将姓名（name）、性别（sex）、电话（telphone）写入缓冲输出流 outbuff 中。

（9）将姓名（name）、性别（sex）、电话（telphone）输出到客户端。

该程序的源代码如下。

<! -- 例程 8-11 diaoyong.java -->

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class diaoyong extends  HttpServlet
{
   FileWriter ou = null;
   BufferedWriter outbuff = null;
   public void init(ServletConfig config) throws ServletException
   {
      super.init(config);
      try{
            ou = new FileWriter("E:\phone.txt",true);
            outbuff = new BufferedWriter(ou);
         }
      catch(IOException ioe)
         {
         }
   }
   public void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
   {
      response.setContentType("text/html;charset = GB2312");
      PrintWriter out = response.getWriter();
      request.setCharacterEncoding("GB2312");
      String name = request.getParameter("name");
      String sex = request.getParameter("sex");
      String telphone = request.getParameter("telphone");
      outbuff.write(name + "\n");
```

```
        outbuff.write(sex + "\n");
        outbuff.write(telphone + "\n");
        outbuff.flush();
        out.println("<html>");
        out.println("<body>");
        out.println("姓名" + name + "<br>");
        out.println("性别" + sex + "<br>");
        out.println("电话" + telphone + "<br>");
        out.println("</body>");
        out.println("</html>");
        out.close();
      }
    public void destroy()
    {
        super.destroy();
        try
           {
             ou.close();
             outbuff.close();
           }
        catch(Exception e)
           {
              e.printStackTrace();
           }
      }
}
```

本例源代码存放于本书配套素材中的 diaoyong.jsp 及 diaoyong.java 文件中(文件路径为 code\8\上机指导)。

8.6.2 留言板

1. 练习目标

(1) 熟练掌握 JSP 页面、Servlet 类、JavaBean 类的创建。

(2) 熟练掌握 Servlet 程序 Web.xml 文件的配置。

(3) 熟悉 JSP+Servlet+JavaBean 程序的运行。

2. 练习指导

JSP+Servlet+JavaBean 的留言板界面如图 8-13 所示。单击"提交留言"按钮后,要将留言人输入的信息保存到数据库中。因此建立一个 JSP 数据库,并建立一个 messages 表来存放留言人输入的信息。messages 表的结构如图 8-14 所示。

列名	数据类型	长度	允许空
id	int	4	
name	varchar	20	✓
email	varchar	20	✓
title	varchar	50	✓
content	text	16	✓

图 8-13 数据库表 messages 的结构

图 8-14 messages. html 用户界面

1）创建填写留言的界面 messages. html

填写留言界面的示例程序为 messages. html，它的执行效果如图 8-14 所示。单击留言界面的“提交留言”按钮，使用 Servlet（AddMessageServlet）接收 HTTP 请求。messages. html 程序源代码如下。

```
<! -- 例程 8 - 12 messages.html -->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>messages.html</title>
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="this is my page">
    <meta http-equiv="content-type" content="text/html; charset=gb2312">
    <!--<link rel="stylesheet" type="text/css" href="./styles.css">-->
  </head>
  <body>
<h2 align="center">留言板</h2>
<hr><form action="/servlet/AddMessageServlet" method="post">
<table width="320" border="1" align="center" cellpadding="0" cellspacing="0">
  <tr>
    <td width="80">姓名:</td>
    <td><input name="name" type="text" id="name" size="25" /></td>
  </tr>
  <tr>
    <td>Email:</td>
    <td><input name="email" type="text" id="email" size="25" /></td>
  </tr>
  <tr>
    <td>主题:</td>
    <td><input name="title" type="text" id="title" size="25" /></td>
  </tr>
  <tr>
    <td>内容:</td>
    <td><textarea name="content" cols=25 rows=7 id="content"></textarea></td>
```

```
  </tr>
  <tr>
    <td colspan="2" align="center">
        <input type="submit" name="Submit" value="提交" />  
        <input name="reset" type="reset" id="reset" value="重置" />
    </td>
  </tr>
</table>
</form>
  </body>
</html>
```

2）创建接受请求保存留言的 Servlet 类

该 Servlet(AddMessageServlet)作为控制器，完成如下工作：

- 接受浏览器发送的所有请求。
- 建立与数据库的连接。
- 将留言板中需要存入数据库的信息存入数据库。
- 对于留言板中“查看留言”的请求，它通过访问另一个 Servlet(ViewMessagesServlet)响应用户的请求。

AddMessageServlet.java 程序源代码如下。

<!-- 例程 8-13 AddMessageServlet.java -->

```
package servlet;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class AddMessageServlet extends HttpServlet {
  //声明数据库连接对象 con
  Connection con = null;
  public AddMessageServlet() {
    super();
  }
  public void destroy() {
    super.destroy();
  }
  public void doGet(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     byte b1[] = request.getParameter("name").getBytes("ISO-8859-1");
```

```
        String name = new String(b1);
        //String name = "chen";
        byte b2[] = request.getParameter("email").getBytes("ISO-8859-1");
        String email = new String(b2);
        //String email = "cqjxnc@163.com";
        byte b3[] = request.getParameter("title").getBytes("ISO-8859-1");
        String title = new String(b3);
        //String title = "通过 servlet 访问数据库";
        byte b4[] = request.getParameter("content").getBytes("ISO-8859-1");
        String content = new String(b4);
        //String content = "通过 servlet 访问数据库";
        if(name == null)   name = "";
        if(email == null)   email = "";
        if(title == null)   title = "";
        if(content == null)   content = "";
        try{
            //将获得的留言信息装入数据库
            PreparedStatement stmt = con.prepareStatement("insert into messages(name,email,
title,content) values(?,?,?,?)");
            stmt.setString(1,name);
            stmt.setString(2,email);
            stmt.setString(3,title);
            stmt.setString(4,content);
            try {
                stmt.execute();
                out.println("数据添加成功!");
                stmt.close();
                con.close();
            }
            catch(Exception e){
                e.printStackTrace();
            }
            //对留言板中"查看留言"的请求
            RequestDispatcher requestDispatcher = request.getRequestDispatcher("/servlet/
ViewMessageServlet");
            requestDispatcher.forward(request,response);
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request,response);
    }
    public void init() throws ServletException {
        String drivername = "com.microsoft.jdbc.sqlserver.SQLServerDriver";
        String url = "jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=jsp";
        String username = "sa";
        String password = "";
        try{
```

```
            //加载 JDBC 驱动程序
            Class.forName(drivername);
            //连接数据库
            con = DriverManager.getConnection(url,username,password);
        }catch(ClassNotFoundException e){
            e.printStackTrace();
        }catch(SQLException e){
            e.printStackTrace();
        }
    }
}
```

【说明】

requestDispatcher = request.getRequestDispatcher("ViewMessages_servlet")语句表示生成一个 RequestDispatcher 类的对象，并将第一个 Servlet 的控制权转交给第二个 Servlet(ViewMessages_servlet)。requestDispatcher.forward(request,response)语句表示服务器端的重定向方式。

RequestDispatcher 是一个 Web 资源的包装器，可以用来把当前的 request 传递到该资源，或者把新的资源包括到当前响应中。RequestDispatcher 的 forward()方法将当前的 request 和 response 重定向到该 RequestDispatcher 指定的资源，它只在服务器端起作用。使用 forward()方法时，Servletengine 传递 HTTP 请求从当前的 Servlet 或 JSP 到另外一个 Servlet、JSP 或普通的 HTML 文件。因为完成一个逻辑操作往往需要跨越多个步骤，每一步骤完成相应的处理后转向到下一个步骤，所以，这种方式在实际项目中大量使用。

使用时应该注意，只有在尚未向客户端输出响应时才可以调用 forward()方法；调用 forward()方法时，如果页面缓存不为空，则在转向前将自动清除缓存，否则将抛出一个 IllegalStateException 异常。

3）创建表示留言数据的 JavaBean 类

MessageDataBean.java 程序建立表示留言板信息的 JavaBean，Bean 中定义的每个属性对应 MessageTable 表中的一个字段，每个属性定义了 getXy()和 setXy()方法，其目的是为显示留言的 JSP 读取 JavaBean 中的数据提供方便。

MessageDataBean.java 程序源代码如下。

<!-- 例程 8-14 MessageDataBean.java -->

```
package servlet;
public class MessageDataBean {
    private String name;
    private String email;
    private String title;
    private String content;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
```

```
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getContent() {
        return content;
    }
    public void setContent(String content) {
        this.content = content;
    }
}
```

4）创建显示留言消息的 JSP 页面

显示留言消息的 JSP 从 JavaBean 中读取留言信息，并且对获得的结果进行显示。viewmessages.jsp 程序源代码如下。

```
<!-- 例程 8-15 viewmessages.jsp -->

<%@ page language="java" import="java.util.*,servlet.MessageDataBean" pageEncoding
="gb2312"%>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+":"+
request.getServerPort()+path+"/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">
    <title>显示留言</title>
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
 </head>
  <body>
      <h2 align="center">所有留言信息</h2>
      <hr>
      <br>
      <table width="80%" align="center" border="1" bgcolor="yellow" cellspacing="0">
```

```
<%
int message_count = 0;
Collection messages = (Collection)request.getAttribute("messages");
Iterator it = messages.iterator();
while(it.hasNext())
{
    MessageDataBean mg = (MessageDataBean)it.next();
%>
      <tr>
          <td width = "25 % ">留言人姓名: </td>
          <td width = "30 % "><% = mg.getName() %></td>
          <td width = "20 % "> Email: </td>
          <td width = "25 % "><% = mg.getEmail() %></td>
      </tr>
      <tr>
          <td>主题: </td>
          <td colspan = "3" width = "80 % "><% = mg.getTitle() %></td>
      </tr>
      <tr>
          <td>留言内容: </td>
          <td colspan = "3"><% = mg.getContent() %></td>
      </tr>
<%
    message_count ++ ;
}
%>
  </table>
  <p align = "center"><a href = "messages.html">我要留言</a></p>
 </body>
</html>
```

5）创建查看留言的 Servlet 类

该 Servlet 作为控制器，完成如下工作：

- 接受 AddMessageServlet 请求。
- 建立与数据库的连接。
- 从数据库中读取存入留言板的信息。这些信息正是留言板界面中"查看留言"按钮所提供的信息。
- 将留言信息提交给 JavaBean（MessageDataBean），再由 JavaBean 对象保留到 Collection 对象中。具体内容见下面的"表示留言板数据的 JavaBean"。
- 把 Collection 对象保存到 request 中，然后访问显示留言的 JSP（viewMessages.jsp）页面。

ViewMessagesServlet.java 的程序源代码如下。

```
<! -- 例程 8 - 16 viewmessages.java -->

package servlet;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
```

```
import java.util.ArrayList;
import java.util.Collection;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class ViewMessageServlet extends HttpServlet {
    //声明数据库连接对象 con
    Connection con = null;
    public ViewMessageServlet() {
        String drivername = "com.microsoft.jdbc.sqlserver.SQLServerDriver";
        String url = "jdbc:microsoft:sqlserver://localhost:1433;DatabaseName = jsp";
        String username = "sa";
        String password = "";
        try {
            Class.forName(drivername);          //加载 JDBC - ODBC 桥驱动程序
            //连接数据库 URL
            con = DriverManager.getConnection(url,username,password);
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
    public void destroy() {
        super.destroy();
    }
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        Collection ret = new ArrayList();
        try{
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("select count( * ) from messages");
            int message_count = 0;
            if(rs.next()){
                message_count = rs.getInt(1);
                rs.close();
            }
            if(message_count > 0){
                rs = stmt.executeQuery("select * from messages");
                while(rs.next()){
                    String name = rs.getString("name");
                    String email = rs.getString("email");
                    String title = rs.getString("title");
                    String content = rs.getString("content");
                    //将数据保存到 MessageDataBean 中
                    MessageDataBean message = new MessageDataBean();
                    message.setName(name);
                    message.setEmail(email);
                    message.setTitle(title);
                    message.setContent(content);
```

```
                    //将 message 对象中的数据取出添加到 ret 对象中
                    ret.add(message);
                }
                rs.close();
                stmt.close();
            }
            //设置名字为 messages 的 request 参数的值,该值由 ret 指定
            request.setAttribute("messages",ret);
            //访问显示留言的 JSP
            RequestDispatcher requestDispatcher = request.getRequestDispatcher
("/viewmessages.jsp");
            requestDispatcher.forward(request,response);
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request,response);
    }
    public void init() throws ServletException {
            }
}
```

本例源代码存放于本书配套素材中的相应文件中(文件路径为 code\8\上机指导)。

本章小结

本章主要介绍了 Servlet 技术的工作原理、基本结构及生命周期等相关概念。Servlet 是指服务器端小程序,主要用于处理客户端传来的 HTTP 请求,并返回一个响应。理解 Servlet 很重要,因为它们是 JSP 的底层实现。

习题 8

一、简答题

1. 简述 Servlet 的基本功能。
2. 简述 Servlet 的工作原理。
3. Servlet 的生命周期包括哪几部分?说明每一部分的具体功能。
4. 一般在什么情况下应该使用 doGet()方法?

二、操作编程题

1. 编写一个在<FORM>标记中访问 Servlet 的加法器程序。要求在 HTML 页面中输入被加数和加数,访问 Servlet 后输出和。
2. 编写一个 JSP 页面程序,读取表单信息。其中,读取表单信息用 Servlet 实现。

第9章 Web开发框架

传统的 Web 应用开发工具(如 ASP、JSP)将页面显示、商业逻辑和数据处理大部分都集中在页面代码中,Web 应用扩展维护比较困难,也不利于开发人员分工协作,尤其在大型 Web 应用开发中愈发显得力不从心。MVC(Model-View-Controller)设计模式将页面显示、商业逻辑和数据处理相分离,最大限度地降低系统各部分之间的耦合性,从而增强系统的可扩展维护性,提高代码重用率,使系统的层次结构更加清晰,易于分工协作。本章讲解如何规划 Web 应用程序的架构上选择一个合适的框架,并重点介绍 Struts 的特点、工作原理及 Struts 的编程技术。

本章主要内容:

- 框架;
- MVC 设计模式;
- JSP 的 Model 1 与 Model 2;
- Struts 框架;
- Struts 开发实例。

9.1 框架概述

伴随着软件开发的发展,在多层的软件开发项目中,可重用、易扩展的且经过良好测试的软件组件,越来越为人们所青睐,这意味着人们可以将充裕的时间用在分析、构建业务逻辑的应用上,而非繁杂的代码工程,于是人们将相同类型的问题解决途径进行抽象,抽取成一个个的框架。

9.1.1 什么是框架

框架,即 Framework,其实就是某种应用的半成品,是一组组件,供选用完成自己的系统。简单地说就是使用别人搭好的舞台来表演。而且,框架一般是成熟的,不断升级的软件,它是结构和创造力之间的一个精确的天平。Web 应用程序框架则鼓励开发人员使用一系列框架所提供的基础类和类库,从而避免杂乱的 JSP 所造成的混乱。

Framework 的体系提供了一套明确机制,从而让开发人员很容易地扩展和控制整个 Framework 开发上的结构。简而言之,Framework 就是易于扩展和控制、能提高开发效率的程序框架。

采用框架技术进行软件开发的主要特点如下。

(1) 领域内的软件结构一致性好。

(2) 建立更加开放的系统。

(3) 重用代码大大增加,软件生产效率和质量也得到了提高。

(4) 软件设计人员要专注于对领域的了解,使需求分析更充分。

(5) 存储了经验,可以让那些经验丰富的人员去设计框架和领域构件,而不必限于低层编程。

(6) 允许采用快速原型技术。

(7) 有利于在一个项目内多人协同工作。

(8) 大量的重用使得平均开发费用降低,开发速度加快,开发人员减少,维护费用降低,而参数化框架使得适应性、灵活性增强。

9.1.2 MVC 设计模式

MVC 模式是 Model-View-Controller 的缩写,中文名称为"模式-视图-控制器",是 20 世纪 80 年代出现的一种软件设计模式,现在已经被广泛地应用在多种应用软件的开发中。它强制地把一个应用的输入、处理、输出流程按照视图(View)、模型(Model)、控制器(Controller)的方式进行分离,3 个核心模块分别承担了不同的任务。

1. 视图

视图是应用程序中用户界面相关的部分,代表用户交互界面,视图向用户显示数据,并能接收用户的输入数据。对于 Web 应用来说,可以概括为 HTML 界面,但有可能为 XHTML、XML 和 Applet。随着应用的复杂性和规模性,界面的处理也变得具有挑战性。一个应用可能有很多不同的视图,MVC 设计模式对于视图的处理仅限于视图上数据的采集和处理,以及用户的请求,而不包括在视图上的业务流程的处理。业务流程的处理交予模型(Model)处理。

2. 模型

模型是应用程序的主体部分,就是业务流程/状态的处理以及业务规则的制定。模型表示业务数据和业务逻辑,一个模型可以为多个视图提供数据,提高了应用的可重用性。业务流程的处理过程对其他层来说是黑箱操作,模型接受视图请求的数据,并返回最终的处理结果。业务模型的设计可以说是 MVC 最主要的核心。业务模型还有一个很重要的模型就是数据模型。数据模型主要指实体对象的数据保存(持续化)。

3. 控制器

控制器工作就是根据用户请求,调用相应的模型组件进行处理,然后调用相应的视图显示模型返回的数据。

一个模型可能对应多个视图,一个视图可能对应多个模型。三者之间的关系如图 9-1 所示。

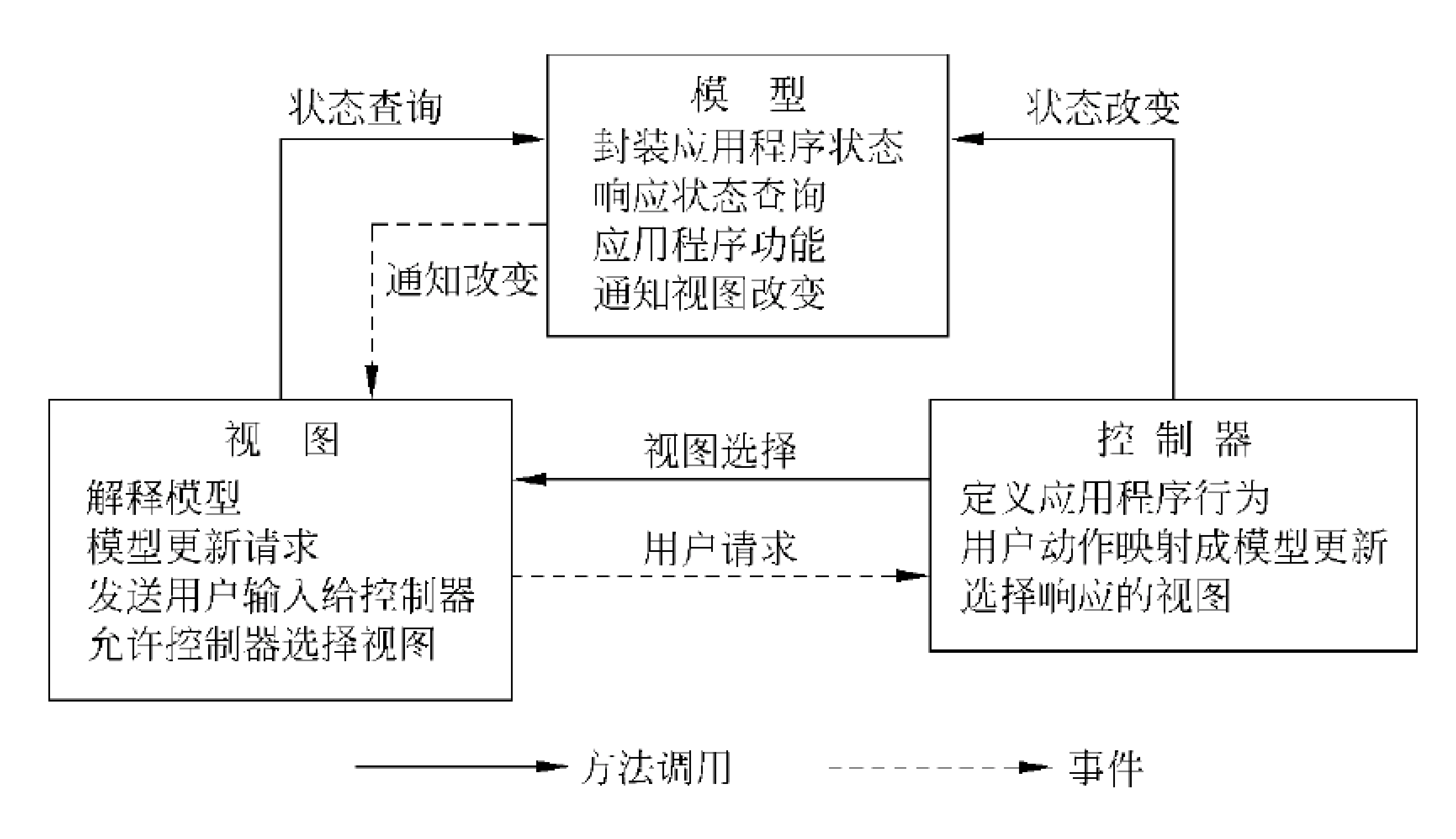


图 9-1 MVC 组件类型的关系和功能

MVC 的出现不仅实现了功能模块和显示模块的分离，同时还提高了应用系统的可维护性、可扩展性、可移植性和组件的可复用性。

9.1.3 JSP 的 Model 1 与 Model 2

尽管 MVC 设计模式很早就出现了，但在 Web 应用的开发中引入 MVC 一直难以实现。在早期的 Java Web 开发应用中，JSP 文件负责业务逻辑、控制网页流程并创建 HTML，这给 Web 开发带来了强耦合、调试困难、程序处理逻辑复杂等问题。为了解决这些问题，Sun 公司在 JSP 出现早期制定了两种规范，分别称为 Model 1 和 Model 2。虽然 Model 1 在一定程度上实现了 MVC，但是它的应用并不尽如人意，直到 Model 2 问世才得以改观。JSP Model 1 和 JSP Model 2 的本质区别在于处理批量请求的位置不同。

1. JSP Model 1 体系结构

JSP Model 1 体系结构如图 9-2 所示，JSP 页面独自响应请求并将处理结果返回客户。这里仍然存在表达与内容的分离，因为所有的数据存取都是由 Bean 来完成的。尽管 Model 1 体系十分适合简单应用的需要，却不能满足复杂的大型应用程序的实现。不加选择地随意运用 Model 1，会导致 JSP 页内被嵌入大量的脚本片段或 Java 代码，特别是当需要处理的请求量很大时，情况更为严重。尽管这对于 Java 程序员来说可能不是什么大问题，但如果 JSP 页面是由网页设计人员开发并维护的（通常这是开发大型项目的规范），这就确实是个问题了。从根本上讲，将导致角色定义不清和职责分配不明，给项目管理带来不必要的麻烦。

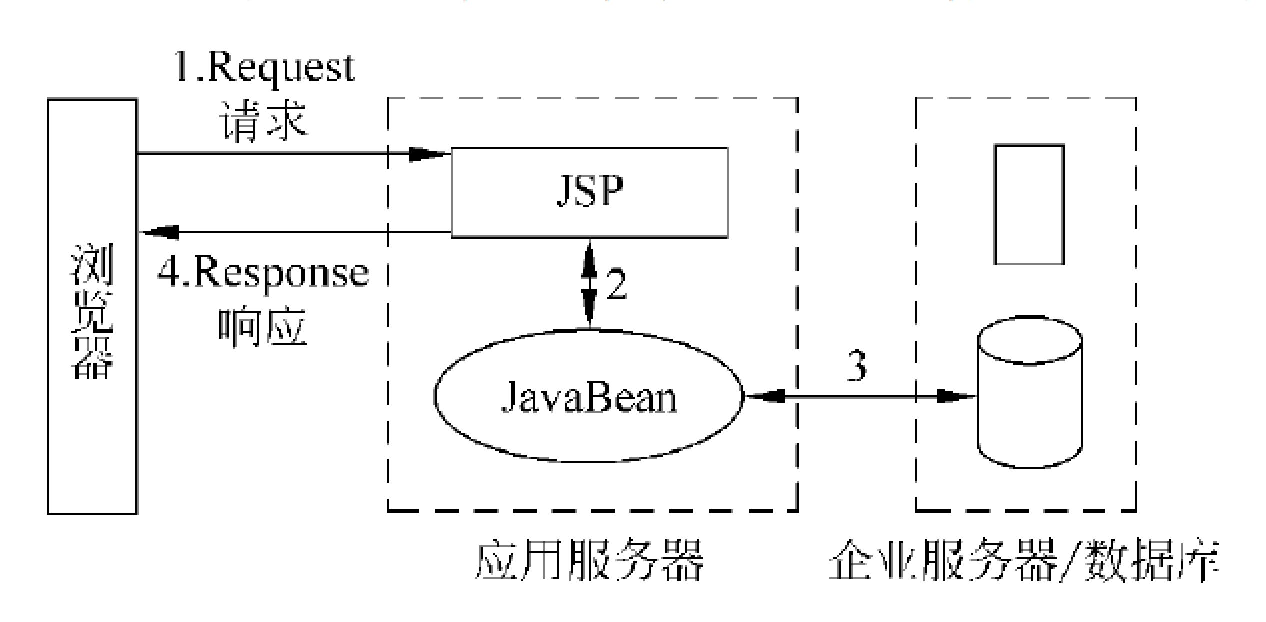


图 9-2 JSP Model 1 体系结构

JSP Model 1 这种架构模型的优点是非常适合小型 Web 项目的快速开发，而且对 Java Web 开发人员的技术水平要求不高。

JSP Model 1 的缺点如下：

- HTML 和 Java 强耦合在一起，导致页面设计与逻辑处理无法分离。
- 可读性差，调试困难，不利于维护。
- 功能划分不清。

2. JSP Model 2 体系结构

JSP Model 2 就是 MVC 架构模式，其体系结构如图 9-3 所示，是一种把 JSP 与 Servlets 联合使用来实现动态内容服务的方法。它吸取了两种技术各自的突出优点，用 JSP 生成表达层的内容，让 Servlets 完成深层次的处理任务。这里 Servlets 充当控制者的角色，负责管理对请求的处理，创建 JSP 页面需要使用的 Bean 和对象，同时根据用户的动作决定把哪个 JSP 页面传给请求者。特别要注意，在 JSP 页内没有处理逻辑；它仅负责检索原先由 Servlets 创建的对象或 Beans，从 Servlet 中提取动态内容插入静态模板，这是一种有代表性的方法，它清晰地分离了表达和内容，明确了角色的定义以及开发者与网页设计者的分工。事实上，项目越复杂，使用 Model 2 体系结构的好处就越大。

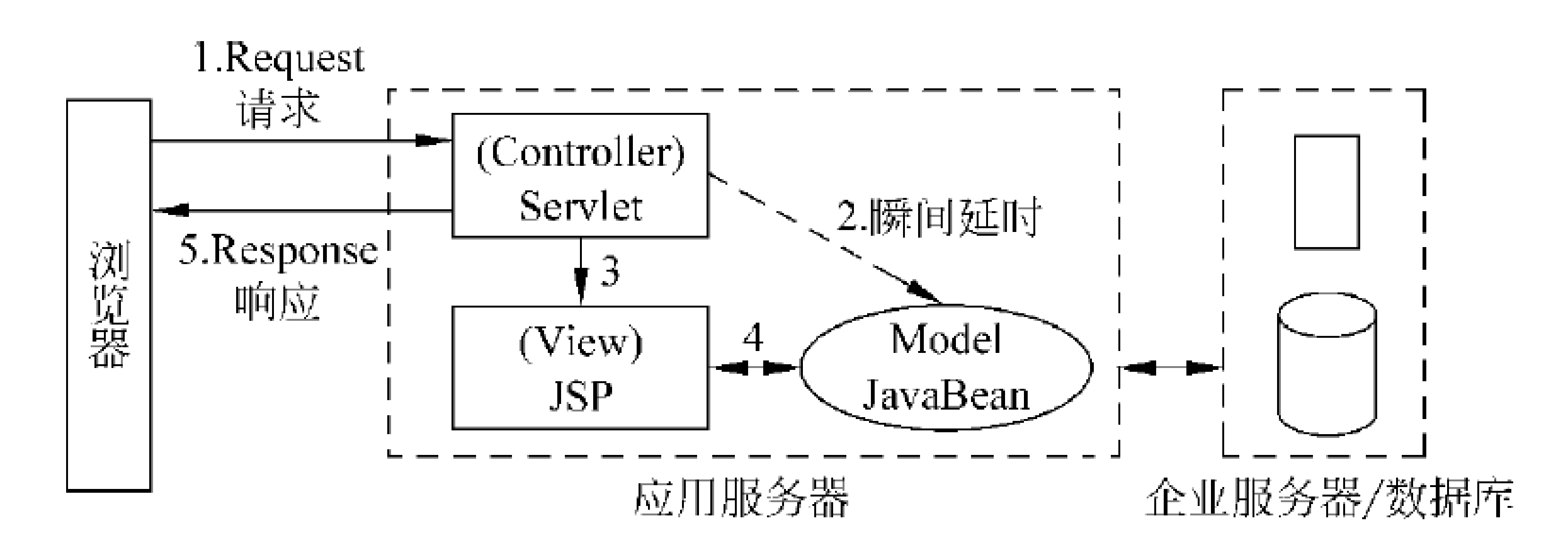


图 9-3 JSP Model 2 体系结构

在 JSP Model 2 中使用了 JSP、Servlet 和 JavaBeans 3 种技术，各技术功能如下所述。

- JSP 负责生成动态网页，只用做显示页面。
- Servlet 负责流程控制，用来处理各种请求的分派。
- JavaBeans 负责业务逻辑、对数据库的操作。

使用 JSP Model 2 的交互过程是，用户通过浏览器向 Web 应用中的 Servlet 发送请求，Servlet 接受到请求后实例化 JavaBeans 对象，调用 JavaBeans 对象的方法，JavaBeans 对象返回从数据库中读取的数据。Servlet 选择合适 JSP，并且把从数据库中读取的数据通过 JSP 进行显示，最后 JSP 页面把最终的结果返回给浏览器。

JSP Model 2 的优点如下：

- 消除了 JSP Model 1 的缺点。
- 该模式适合多人合作开发大型的 Web 项目。
- 各司其职，互不干涉。
- 有利于开发中的分工。
- 有利于组件的重用。

JSP Model 2 的缺点是 Web 项目的开发难度加大，同时对开发人员的技术要求也提高了。

9.2 Struts 框架

Struts 框架是一个由 Apache 软件基金会发起的一个开源项目，实质上就是在 JSP Model 2 的基础上实现的一个 MVC 框架。Struts 这个名字来源于在建筑和旧式飞机中使用的支持金属架，它的目的是帮助开发人员减少运用 MVC 设计模型来开发 Web 应用的时间。

Struts 工程设计的意图是为 JSP 的 Web 应用开发提供一个易于把显示层和事务层分离的开源框架，在这个框架支持下很容易实现 JSP 系统的 Model 2 开发模式。Struts 框架具有可靠性高、适应性强、开发和维护时间短和易于维护等优点。

9.2.1 Struts 的基本结构

Struts 是基于 MVC 的 Web 应用框架，由一组相互协作的类、Servlet 以及 JSP TagLib 组成。在 Struts 框架中，模型由实现业务逻辑的 JavaBean 或 EJB 组件构成，控制器由 ActionServlet 和 Action 来实现，视图由一组 JSP 文件构成。Struts 框架的体系结构如图 9-4 所示。

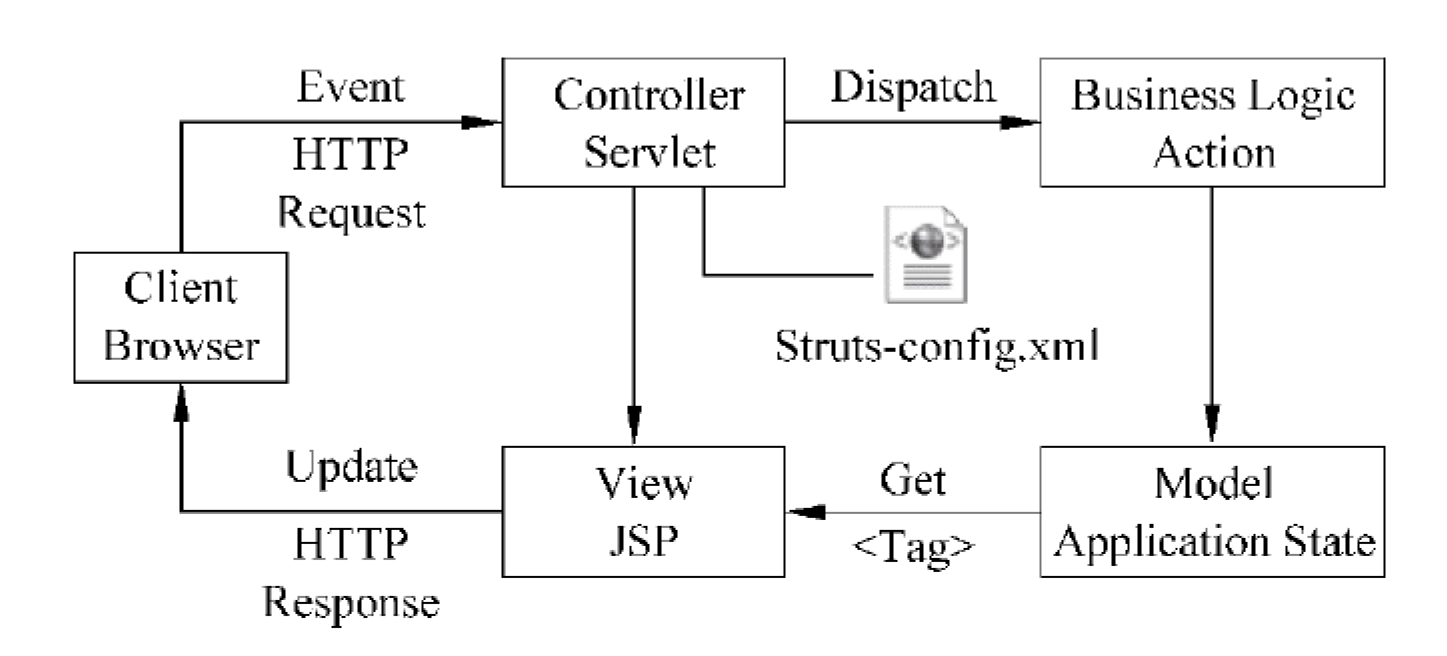


图 9-4 Struts 框架的体系结构

(1) 视图：主要是通过 JSP 技术生成页面完成视图，利用 Struts 提供的自定义标记库编写用户界面，将应用逻辑和显示逻辑进行分离。Struts 框架通过这些自定义标记建立了 View 和 Model 之间的联系，可以实现和 Model 部分中的 ActionForm 的映像，完成对用户数据的封装。

(2) 控制器：作用是从客户端接受请求，并选择执行相应的业务逻辑，然后把响应结果送回到客户端。在 Struts 中，控制器由 ActionServlet 对象和 ActionMapping 对象构成，核心是一个 Servlet 类型的 ActionServlet 对象，它用来接受客户端的请求并分发到相应的动作类(Action 类的子类)。ActionServlet 对象包括一组基于配置的 ActionMapping 对象，每个 ActionMapping 对象实现了一个请求到一个具体的 Model 部分中 Action 对象之间的映射。

(3) 模型：在 Struts 中，Model 可以分为系统的内部状态和改变系统状态的行为(业务逻辑)两部分。系统的内部状态常由一组 JavaBean 表示，业务逻辑由 Action 对象和 ActionForm 的类对象进行处理。Action 对象封装了具体的处理逻辑，调用业务逻辑模块，并且把响应提交给合适的 View 以产生响应业务对象更新应用程序的状态。ActionForm 对象可以派生子类对象，通过结合自定义标记库来实现对客户端的表单数据的良好封装和支持。Action 对象可以直接对 ActionForm 对象进行读写，而不再需要与 request 对象和 response 对象进行数据交互。通过 ActionForm 对象实现了对 View 和 Model 之间交互的支持。

(4) Struts-config. xml：用于建立 Controller 和 Model 之间的关系，将各部分紧密联系在一起。它描述了 Controller 所使用的把请求对应到具体处理的法则，同时描述了客户提供的数据与 ActionForm 组件的对应映射关系。

9.2.2 Struts 的工作流程

对于采用 Struts 框架的 Web 应用，在 Web 应用启动时就会加载并初始化 ActionServlet，ActionServlet 从 struts-config. xml 文件中读取配置信息，然后把它们存放到各种配置对象中，如 Action 对象的映射信息存放在 ActionMapping 对象中。

客户端(Client)通过本地浏览器(Browser)向服务器发出一个请求(HTTP Request)，当 ActionServlet 接收到一个客户请求时，其具体工作流程如下：

(1) 检索和用户请求匹配的 ActionServlet 实例，如果不存在，就返回用户请求路径无效的信息。

(2) 如果 ActionForm 实例不存在，就创建一个 ActionForm 对象，把客户提交的表单数据保存到 ActionForm 对象中。

(3) 根据配置信息决定是否需要表单验证。如果需要验证，就调用 ActionForm 的 validate()方法。

(4) 如果 ActionForm 的 validate()方法返回 null 或返回一个不包含 ActionMessage 的 ActionErrors 对象，就表示表单验证成功。

(5) ActionServlet 根据 ActionMapping 实例包含的映射信息决定将请求转发给哪个 Action。如果相应的 Action 实例不存在，就先创建这个实例，然后调用 Action 的 execute()方法。

(6) Action 的 execute()方法返回一个 ActionForward 对象，ActionServlet 再把客户请求转发给 ActionForward 对象指向的 JSP 组件。

(7) ActionForward 对象指向的 JSP 组件生成动态网页，并返回客户。

对于流程(4)，如果 ActionForm 的 validate()方法返回一个包含一个或多个 ActionMessage 的 ActionErrors 对象，就表示表单验证失败，此时 ActionServlet 将直接把请求转发给包含用户提交表单的 JSP 组件。在这种情况下，不会再创建 Action 对象并调用 Action 的 execute()方法。

显示了 Struts 响应用户请求的工作流程如图 9-5 所示。

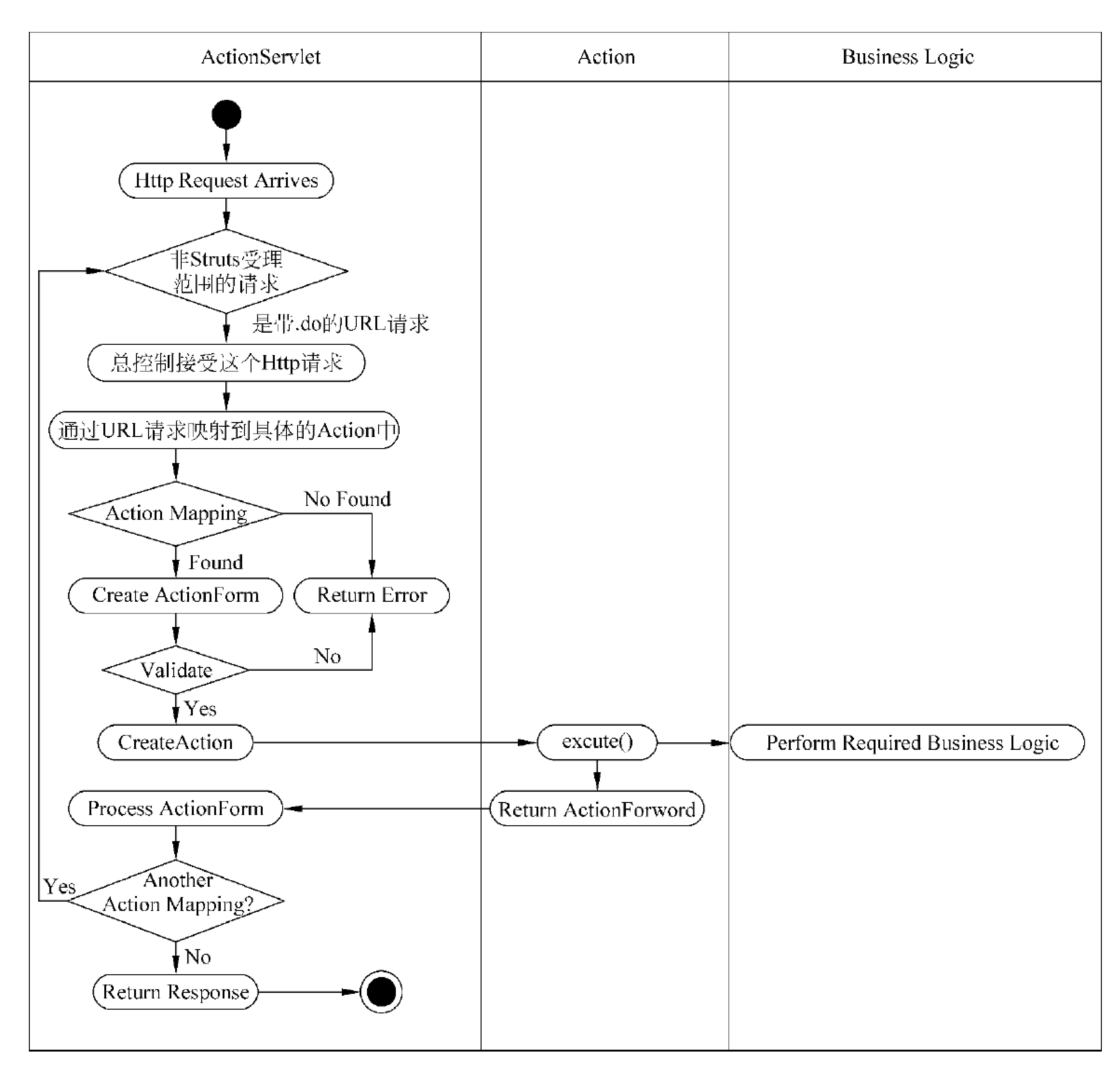


图 9-5　Struts 的工作流程

9.2.3　Struts 的组件

Struts 的 4 个核心组件有 ActionServlet、Action、ActionMapping（包含 ActionForward）和 ActionForm 组件，这些组件的作用主要是控制和处理客户的请求。

1. ActionServlet 控制器组件

ActionServlet 继承自 javax. servlet. http. HttpServlet 类，其在 Struts framework 中扮演的角色是中心控制器。它提供一个中心位置来处理全部的终端请求。控制器 ActionServlet 主要负责将 HTTP 的客户请求信息组装后，根据配置文件的指定描述转发到适当的处理器。

按照 Servlet 的标准，所有的 Servlet 必须在 Web 配置文件中（web. xml）声明。同样，ActionServlet 必须在 Web Application 配置文件（web. xml）中描述，有关配置信息如下：

```
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
</servlet>
```

全部的请求 URL 以 *.do 的模式存在并映射到 servlet，其配置如下：

```
<servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

一个该模式的请求 URL 符合如下格式：

```
http://www.my_site_name.com/mycontext/actionName.do
```

中心控制器为所有的表示层请求提供了一个集中的访问点。这个控制器提供的抽象概念减轻了开发者建立公共应用系统服务的困难，如管理视图、会话及表单数据。它也提供一个通用机制，如错误及异常处理、导航、国际化、数据验证、数据转换等。

当用户向服务器端提交请求时，实际上信息是首先发送到控制器 ActionServlet，一旦控制器获得了请求，就会将请求信息传交给一些辅助类(help classes)进行处理。这些辅助类知道如何去处理与请求信息所对应的业务操作。在 Struts 中，这个辅助类就是 org.apache.struts.action.Action。通常开发者需要自己继承 Action 类，从而实现自己的 Action 实例。

2. Action 行为组件

ActionServlet 把全部提交的请求都被控制器委托到 RequestProcessor 对象。RequestProcessor 使用 struts-config.xml 文件检查请求 URL 找到动作 Action 标示符。

一个 Action 类的角色，就像客户请求动作和业务逻辑处理之间的一个适配器(Adaptor)，其功能就是将请求与业务逻辑分开。这样的分离使得客户请求和 Action 类之间可以有多个点对点的映射。而且 Action 类通常还提供了其他的辅助功能，比如认证(Authorization)、日志(Logging)和数据验证(Validation)。

Action 最为常用的是 execute()方法，代码如下。

```
public ActionForward execute(ActionMapping mapping,
                             ActionForm form,
                             javax.servlet.ServletRequest request,
                             javax.servlet.ServletResponse response)
                     throws java.io.IOException,javax.servlet.ServletException
```

当 Controller 收到客户的请求，再将请求转移到一个 Action 实例时，如果这个实例不存在，控制器会首先创建实例，然后调用 Action 实例的 execute()方法。Struts Framework 为应用系统中的每一个 Action 类只创建一个实例。因为所有的用户都使用这一个实例，所以必须确定 Action 类运行在一个多线程的环境中。

专家点拨：客户自己继承的 Action 子类必须重写 execute()方法，因为 Action 类在默认情况下是返回 null 的。

3. ActionMapping 映射组件

上面讲到了一个客户请求是如何被控制器转发和处理的，但是，控制器如何知道什么样的信息转发到什么样的 Action 类呢？这就需要一些与动作和请求信息相对应的映射配置来说明。在 Struts 中，这些配置映射信息存储在特定的 XML 文件中（比如 struts-config.xml）。

这些配置信息在系统启动时被读入内存，供 Struts framework 在运行期间使用。而在内存中，每一个<action>元素都与 org.apache.struts.action.ActionMapping 类的一个实例对应。下面代码就显示了一个登录的配置映射。

```
<action-mappings>
    <action   path="/loginAction"
            type="com.test.LoginAction"
            name="LoginForm"
            scope="request"
            input="logincheck.jsp"
              validate="false">
        <forward name="welcome" path="/welcome.jsp"/>
        <forward name="failure" path="/login_failure.jsp "/>
    </action>
</action-mappings>
```

上面的配置表示当可以通过/loginAction.do（此处假设配置的控制器映射为 *.do）提交请求信息时，控制器将信息委托 com.test.LoginAction 进行处理，调用 LoginAction 实例的 execute()方法，同时将 Mapping 实例和所对应的 LoginForm 信息传入。其中 name=LoginForm，使用的是 form-bean 元素所声明的 ActionForm。有关 Form-bean 的声明如下代码所示。

```
<form-beans>
  <form-bean   name="LoginForm" type="com.test.LoginForm"/>
</form-beans>
```

4. ActionForward 导航组件

ActionForward 对象是配置对象。这些配置对象拥有独一无二的标识以允许它们按照有意义的名称如 success、failure 等来检索。ActionForward 对象封装了向前进的 URL 路径且被请求处理器用于识别目标视图。ActionForward 对象建立自<forward>元素位于 struts-config.xml。下面是一个 Struts 中<forward>元素例子，属于<action>元素范围。

```
<action path="/editCustomerProfile"
        type="packageName.EditCustomerProfileAction"
        name="customerProfileForm" scope="request">
   <forward name="success" path="/MainMenu.jsp"/>
   <forward name="failure" path="/CustomerService.jsp"/>
</action>
```

基于执行请求处理器的 execute()方法的结果，当传递一个值匹配指定于<forward>元素中 name 属性的值的时候，下一个视图可以在 execute()方法中被开发者用方便的 org. apache. struts. action. -ActionMapping. findForward()方法选择。ActionMapping. findForward()方法既从它的本地范围又从全局范围提供一个 ActionForward 对象，该对象返回至 RequestProcessor，以 RequestDispatcher. forward()或 response. sendRedirect()方法调用下一个视图。当<forward>元素有 redirect＝false 属性或 redirect 属性不存在时，RequestDispatcher. forward()被执行；当 redirect＝true 时，将调用 sendRedirect()方法。下例举例说明了 redirect 属性的用法：

```
<forward name = "success" path = "/success. jsp" redirect = "true"/>
```

如果 redirect＝true，URL 建立如/contextPath/path，因为 HttpServletResponse. sendRedirect()中解释 URL 采用“/”开头相对于 Servlet 容器根目录。如果 redirect＝false，URL 建立如/path，因为 ServletContext. getRequestDisptacher()采用虚拟目录相关 URL。

在此稍稍介绍一下有关 global-forwards 的概念。其在配置文件中描述了整个应用系统可以使用的 ActionForward，而不仅仅是一个特定的 Action。

```
<global - forwards>
    <forward name = "logout" path = "/logout. do"/>
  <forward name = "error"   path = "/error. jsp"/>
</global - forwards>
```

5. ActionForm 组件

一个应用系统的消息转移(或者说状态转移)的非持久性数据存储，是由 ActionForm 负责保持的。ActionForm 派生的对象用于保存请求对象的参数，因此它们和用户紧密联系。

一个 ActionForm 类被 RequestProcessor 建立后，这时产生一个重定位的 URL，该 URL 是为映射到控制器 Servlet 指定表单属性的，而不是 JSP 和相应的动作映射。在这个情况下，如果没有在指定的活动范围内找到，RequestProcessor 将尝试寻找可能导致创建一个新 ActionForm 对象的表单 Bean。ActionForm 对象在指定的活动范围内被用<action>元素的 name 属性找到。

RequestProcessor 将随后重新安排表单属性，用请求时参数填充表单，随即调用表单对象的 validate()方法以履行服务器端用户输入验证。仅当 ActionMapping 对象中 validate 属性被设为 true 时，validate()方法被调用；这就是默认的行为。request. getParameterValues()方法被用于得到一个 String[]对象，它用来填充表单；验证的结果应该是一个 ActionErrors 对象，用 org. apache. struts. taglib. html. ErrorsTag 来显示验证错误给用户。ActionForm 也可以被用于为当前用户保存即将被一个视图引用的中间模型状态。

当一个表单对象被 RequestProcessor 找到，它被传递到请求处理器的 execute()方法。一个 ActionForm 对象也可以被请求处理器建立。表单对象建立目的是提供中间模型状态给使用请求范围 JSP；这将确保对象不会在有效性过期后仍然存在。默认的，所有的表单都被保存为会话范围。会话中表单对象脱离有效性的存在可能导致浪费内存，同样地，请求处理器必须跟踪保存在会话中的表单对象的生命周期。一个好的捕获表单数据的实践是为

横跨多用户交互的相关表单用一个单独的表单 Bean。表单 Bean 也可以在反馈的时候用来储存能够被自定义标签改变的中间模型状态。在视图中标签用法避免结合 Java 代码，因此要成为一个好的任务划分，Web 生产组主要处理标志，而应用开发组主要处理 Java 代码。标签因素退出访问中间模型状态的逻辑；当访问嵌套的对象或当通过聚集列举时这个逻辑可能很复杂。

对于每一个客户请求，Struts Framework 在处理 ActionForm 的时候，一般需要经历如下几个步骤。

(1) 检查 Action 的映射，确定在 Action 中已经配置了对 ActionForm 的映射。

(2) 根据 name 属性，查找 Form bean 的配置信息。

(3) 检查 Action 的 Form bean 的使用范围，确定在此范围下，是否已经有此 Form bean 的实例。

(4) 假如在当前范围下，已经存在了此 Form bean 的实例，而对当前请求来说，是同一种类型，那么就重用。

(5) 否则，就重新构建一个 Form bean 的实例。

(6) Form bean 的 reset()方法备调用。

(7) 调用对应的 setter()方法，对状态属性赋值。

(8) 如果 validated 的属性被设置为 true，那么就调用 Form bean 的 validate()方法。

(9) 如果 validate()方法没有返回任何错误，控制器将 ActionForm 作为参数，传给 Action 实例的 execute()方法并执行。

专家点拨：直接从 ActionFrom 类继承的 reset()方法和 validate()方法，并不能实现什么处理功能，所以有必要自己重新覆盖。

9.2.4 Struts 的配置文件

Struts 的配置文件包括 struts-config. xml 和 web. xml 文件。

1. struts-config. xml 配置文件

struts-config. xml 配置文件位于应用的 WEB-INF 目录。该文件是 Struts 中的核心文件，该文件配置各种组件文件的配置包括全局转发、ActionMapping 类、ActionForm Bean 和 JDBC 数据源 4 个部分。

1) 配置全局转发

全局转发用来在 JSP 页面之间创建逻辑名称映射。转发都可以通过对调用操作映射的实例来获得。例如：

```
<global - forwards>
    <forward name = "success"  path = "/success.do"/>
    <forward name = "failure"   path = "/failure.jsp"/>
</global - forwards>
```

其中，name 属性用于设置全局转发的名字，path 属性用于设置与目标 URL 的相对路径。

2) 配置 ActionMapping 类

ActionMapping 类帮助进行框架内部的流程控制，它们可以将请求 URL 映射到

Action 类,并且将 Action 类与 ActionForm Bean 相关联。ActionServlet 在内部使用这些映射,并将控制转移到特定 Action 类的实例。所有 Action 类使用 perform()方法来实现特定应用程序代码,返回一个 ActionForward 对象,其中包括响应转发的目标资源名称。例如:

```
<action-mappings>
    <action path="/login" type="LoginAction" name="loginForm" scope="request" input=
"/login.jsp">
    </action>
    <forward name="success" path="/success.jsp"/>
    <forward name=" failure " path="/failure.jsp"/>
</action-mappings>
```

Action 元素信息说明如表 9-1 所示。

表 9-1 <Action>元素配置信息说明

属　　性	描　　述
Path	Action 类的相对路径
Name	与本操作关联的 Action bean 名称
Type	连接到本映射的 Action 类的全称(可有包名)
Scope	Actionform bean 的作用域(请求或会话)
Prefix	用来匹配请求参数与 bean 属性的前缀
Suffix	用来匹配请求参数与 bean 属性的后缀
attribute	作用域名称
ClassName	ActionMapping 对象的类的完全限定名,默认值是 org.apache.struts.action.ActionMapping
input	输入表单的路径,指向 bean 发生输入错误必须返回的控制
unknown	设为 true,操作将被作为所有没有定义的 ActionMapping 的 URL 的默认操作
validate	设置为 true,则在调用 Action 对象上的 perform()方法前,ActionServlet 将调用 Actionform bean 的 validate()方法来进行输入检查

通过<forward>元素可以定义资源的逻辑名称,该资源是 Action 类的响应要转发的目标,如表 9-2 所示。

表 9-2 <forward>元素配置信息说明

属　　性	描　　述
Id	ID
ClassName	ActionForward 类的完全限定名,默认值是 org.apache.struts.action.ActionForward
Name	操作类访问 ActionForward 时所用的逻辑名
Path	响应转发的目标资源的路径
redirect	若设置为 true,则 ActionServlet 使用 sendRedirec 方法来转发资源

3) 配置 ActionForm Bean

ActionServlet 使用 ActionForm Bean 来保存请求的参数,这些 bean 的属性名称与 HTTP 请求参数中的名称相对应,控制器将请求参数传递到 ActionForm Bean 的实例,然后将这个实例传送到 Action 类,其代码如下。

```
<form - beans>
    <form - bean name = "loginForm" type = "LoginForm"/>
</form - beans>
```

<form-beans>元素配置信息如表 9-3 所示。

表 9-3 <form-beans>元素配置信息说明

属性	描述
Id	ID
ClassName	ActionForm Bean 的完全限定名，默认值是 org. apache. struts. action. ActionFormBean
Name	表单 Bean 在相关作用域的名称，这个属性用来将 Bean 与 ActionMapping 进行关联
Type	类的完全限定名

4）配置 JDBC 数据源

配置 JDBC 数据源使用<data-sources>元素可以定义多个数据源，例如：

```
<data - sources>
    <data - source id = "DS1" key = "conPool" type =
        "org.apache.struts.util.GenericDataSource"
        < set - property id = " SP1" autoCommit = " true" description = " Example Data Source
Configuration"
        driverClass = "org.test.mm.mysql.Driver" maxCount = "4"
        minCount = "2" url = "jdbc:mysql://localhost/test" user = "sa" password = "123" />
    <data - source/>
</data - sources>
```

<data-sources>元素配置信息如表 9-4 所示。

表 9-4 <data-sources>元素配置信息说明

属性	描述
desciption	数据源的描述
autoCommit	数据源创建的连接所使用的默认自动更新数据库模式
driverClass	数据源所使用的类
minCount	最小连接数

2. web.xml 配置文件

每一个规范的 Web 应用在 WEB-INF 目录下都应该有一个 web. xml 配置文件，用来对 Web 应用的属性进行配置。web. xml 配置文件包含的内容很多，但是基本的配置应该包括如下几部分：

- 环境参数初始化。
- Servlet 配置。
- 过滤器配置。
- 监听器配置。
- JNDI 配置。

- Session 配置。
- JSP 网页相关配置。
- MIME TYPE 配置。
- welcome 文件清单。
- 错误处理。

1）根元素和头

因为该配置文件是一个 xml 文档，因此必须遵循 xml 的书写规范，例如，大小写敏感，需要声明版本号和字符编码等，一个没有任何配置的原始的 xml 文件代码如下。

```
<?xml version = "1.0" encoding = "ISO - 8859 - 1"?>
<web - app xmlns = "http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi = "http://www.w3.org/2001/XMLSchema - instance"
    xsi:schemaLocation = "http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web - app_2_4.xsd" version = "2.4">
</web - app>
```

2）环境参数初始化

根元素：<init-param>。

子元素：<param-name>和<param-value>，分别对应参数的名称和参数取值，其中全局参数的设置必须在与文档有关的元素之后，而必须在 filter、listener 及 servlet 等元素之前；Servlet 的初始参数在<servlet-class>之后；在 JSP 中初始参数在<jsp-file>之后；filter 初始参数在<filter-class>之后。应用范围内的初始化参数可以通过 ServletContext 的 getInitParameter()方法获得。

```
<! -- 全局范围内环境参数初始化 -->
<context - param>
   <param - name>username</param - name>
   <param - value>admin</param - value>
</context - param>
```

3）Servlet 配置

该配置在 Servlet 简介一文中有相关介绍，在这里就省略了，要增加的说明就是如果一个 Servlet 有多个映射，则需要写出多个<servlet-mapping>模块，而不能在一个<servlet-mapping>模块中写出多个<url-pattern>。

```
<servlet>
  <description>add student</description>
  <display - name>Student_add</display - name>
  <servlet - name>Student_add</servlet - name>
  <servlet - class>servlets.Student_add</servlet - class>
  <init - param>
     <param - name>...</param - name>
     <param - value>...</param - value>
  </init - param>
</servlet>
<! -- servlet 映射名称及映射路径 -->
<servlet - mapping>
```

```
    <servlet-name>Student_add</servlet-name>
    <url-pattern>/student/student_add</url-pattern>
</servlet-mapping>
```

4）过滤器配置

过滤器可以截取和修改一个 Servlet 或 JSP 页面的请求或从一个 Servlet 或 JSP 页面发出的响应，用<filter>元素和<filter-mapping>元素完成设置。

```
<!-- 过滤器定义 -->
 <filter>
    <filter-name>EncodingFilter</filter-name>
    <filter-class>filter.EncodingFilter</filter-class>
    <init-param>
       <param-name>Encoding</param-name>
       <param-value>GB2312</param-value>
    </init-param>
 </filter>
 <!-- 过滤器映射路径，可以与一个或多个 Servlet 或 JSP 页面相关联 -->
<filter-mapping>
     <filter-name>EncodingFilter</filter-name>
     <url-pattern>/*</url-pattern>
</filter-mapping>
```

5）监听器配置

注册一个监听程序用<listener>元素，在 listener 元素中，只有一个<listener-class>子元素，指明监听器对应的类。

```
<!-- 监听器配置 -->
<listener>
  <listener-class>listener.OnlineListener</listener-class>
</listener>
```

6）数据库连接池配置

```
<!-- 数据库连接池配置 -->
<resource-ref>
    <description>DataSource</description><!-- 描述 -->
    <res-ref-name>jdbc/mysql/bookstore</res-ref-name><!-- 资源名称 -->
    <res-type>javax.sql.DataSource</res-type><!-- 资源类型 -->
    <res-auth>Container</res-auth><!-- 指出由 Application 或 Container 提供 -->
</resource-ref>
```

7）Session 配置

如果某个会话在一定时间未被访问，则服务器可以将其扔掉以节约内存，该功能的实现可借助 web.xml 文件来实现，使用<session-config>及其子元素<session-timeout>，其中过期时间以分钟为单位。

```
<!-- 设置会话过期时间 -->
<session-config>
  <session-timeout>180</session-timeout>
</session-config>
```

8）JSP 网页相关配置

<jsp-config>元素主要用来设置 JSP 的相关配置，该元素包括<taglib>和<jsp-property-group>两个子元素，分别完成标签库和 JSP 相关属性的设置。

```
<jsp-config>
  <!-- 自定义标签配置 -->
  <taglib>
      <taglib-uri>/formtag</taglib-uri><!-- 与 tld 中的 uri 一致 -->
      <taglib-location>/WEB-INF/tlds/FormTag.tld</taglib-location><!-- tld 文件路径 -->
  </taglib>
  <!-- 设置 jsp-property-group -->
  <jsp-property-group>
      <display-name>bookstore4</display-name>
      <url-pattern>*.jsp</url-pattern><!-- 设定影响范围 -->
      <el-ignored>false</el-ignored><!-- 是否支持 EL 语法 -->
      <scripting-invalid>false</scripting-invalid> <!-- 是否支持脚本 -->
      <page-encoding>gb2312</pageencoding><!-- 设置编码 -->
      <include-prelude>head.jspf</include-prelude><!-- 设置 jsp 网页的开始 -->
      <include-coda>end.jspf</include-coda> <!-- 设置 jsp 网页的结尾 -->
   </jsp-property-group>
</jsp-config>
```

9）Welcome 文件清单

指出默认的欢迎界面，按照顺序依次查找，直到找到对应的页面为止。

```
<!-- welcome 欢迎清单 -->
<welcome-file-list>
  <welcome-file>index.html</welcome-file>
  <welcome-file>index.htm</welcome-file>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

10）错误处理

主要是当 JSP 页面或 Servlet 出现错误或者抛出异常时，显示指定的页面。

- 根元素：<error-page>。
- 子元素：<error-code>设置 HTTP 错误代码；<exception-type>设置 java 异常类型，<error-location>元素用来设置发生错误或异常时要显示的页面。

```
<!-- 错误处理 -->
<error-page>
  <error-code>404</error-code>
  <location>/error.jsp</location>
</error-page>
<error-page>
  <exception-type>exception.loginerror</exception-type>
  <location>/loginerror.jsp</location>
</error-page>
```

9.3　Struts 开发实例

本节采用 Struts 框架讲解一个简单的 Struts 应用例子——登录系统，通过这个例子来理解 Struts 应用的基本经验。该例子的功能：接受用户输入的用户名<username>和密码<password>，如果用户名和密码正确，则登录成功输出"登录成功!"，否则输出"登录失败!"。

9.3.1　模块构成

Struts 框架可以方便迅速地把一个复杂的应用划分模型、视图和控制器组件，而 Struts 的配置文件 struts-config.xml 则可以灵活地组装这些组件，简化开发过程。以下是该例子应用的各个模块的构成。

- 模型包括一个 JavaBean 组件 User，它有 userName 和 password 属性，代表用户名和密码。它提供了 get()/set()方法，分别用于读取和设置 userName 和 password 属性，也可以提供一些方法，负责把这两个属性保存到持久化存储系统中，例如，数据库或文件系统。对于更为复杂的 Web 应用，JavaBean 组件可以作为 Web 服务的前端组件。
- 视图包括一个 JSP 文件 login.jsp，它提供用户界面，接受用户输入的姓名。视图还包括一个 ActionForm Bean(LoginForm)，它用来存放表单数据并进行表单验证，如果用户没有输入用户名和密码就提交表单，将返回出错信息。
- 控制器包括一个 Action 类 LoginAction，它完成两项任务：一是进行业务逻辑验证，如果用户输入的用户名或密码错误，将返回错误消息；二是决定将合适的视图组件返回给用户。

除了创建模型、视图和控制器组件，还需要创建 Struts 的配置文件 struts-config.xml，它可以把这些组件组装起来，使它们协调工作。此外，还需要创建整个 Web 应用的配置文件 web.xml。

9.3.2　创建模型组件

模型组件包括一个 JavaBean 组件 User，它有 userName 和 password 属性，代表用户名和密码。它提供了 get()/set()方法，分别用于读取和设置 userName 和 password 属性。User.java 源代码如下。

<!-- 例程 9-1 User.java -->

```
package bean;
public class User {
    private String userName;
    private String password;
    public String getUserName() {
        return userName;
    }
```

```
        public void setUserName(String userName) {
            this.userName = userName;
        }
        public String getPassword() {
            return password;
        }
        public void setPassword(String password) {
            this.password = password;
        }
    }
```

9.3.3 创建视图组件

视图包括一个 JSP 文件 login.jsp，它提供用户界面，接受用户输入的姓名。视图还包括一个 ActionForm Bean(LoginForm)，它用来存放表单数据并进行表单验证，如果用户没有输入用户名和密码就提交表单，将返回出错信息。

1. 创建 login.jsp 文件

login.jsp 提供用户界面，能够接受用户输入的用户名和密码，login.jsp 源代码如下所示。

```
<!-- 例程 9-2 login.jsp -->

//声明和加载 Struts 的标签库
<%@ page language="java" pageEncoding="gb2312" %>
<%@ taglib url="http://struts.apache.org/tags-bean" prefix="bean" %>
<%@ taglib url="http://struts.apache.org/tags-html" prefix="html" %>

<html>
    <head>
        <title>登录系统</title>
    </head>
    <body>
        <html:form action="/login">
        <table align="center" border="0" width="400">
            <tr>
            <td colspan="2" align="center">用户登录</td>
            </tr>
            <tr>
            <td align="right" width="80">用户名:</td>
            <td><html:text property="username" maxlength="20" size="20"/>
              <font color="#ff0000">
                <html:errors property="username"/>
              </font>
            </td>
            </tr>
            <tr>
            <td align="right">密  码:</td>
            <td><html:password property="password" maxlength="20" size="20"/>
              <font color="#ff0000">
```

```
                <html:errors property = "password"/>
              <font>
            </td>
            </tr>
            <tr>
            <td colspan = "2" align = "center"><html:submit value = "登录"/>  
              <html:reset value = "重置">
              </html:reset></td>
            </tr>
        </table>
        </html:form>
    </body>
</html>
```

【说明】

程序中

```
<html:form action = "/login">
⋮
<html:text property = "username" maxlength = "20" size = "20"/>
⋮
<html:errors property = "username"/>
```

这些代码都使用了 Struts 的客户化标签,客户化标签是联系视图组件和 Struts 框架中其他组件的纽带,这些标签可以访问或显示来自于控制器和模型组件的数据。

在本程序中仅使用了 HTML 标签库中的标签,Struts HTML 标签可以完成和标准的 HTML 元素相同的功能,但是这些标签可以和 Struts 框架中其他组件产生错误消息。代码<html:errors property="username"/>,显示 LoginForm Bean 中 validate()方法对表单中的数据进行合法性验证时所产生的错误信息。

语句<html:form action="/login">,指定将 form 提交给 login.do。login.do 是一个 Action,在 Struts 的配置文件 struts-config.xml 中进行了说明和定义。

2. 创建 success.jsp 文件

success.jsp 显示用户登录成功的信息,success.jsp 源代码如下所示。

<!-- 例程 9-3 success.jsp -->

```
<%@ page language = "java" import = "java.util.*" pageEncoding = "gb2312" %>
<%
String path = request.getContextPath();
String basePath = request.getScheme() + "://" + request.getServerName() + ":" + request.
getServerPort() + path + "/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href = "<% = basePath %>">
    <title>登录成功</title>
    <meta http-equiv = "pragma" content = "no-cache">
```

```
        <meta http-equiv="cache-control" content="no-cache">
        <meta http-equiv="expires" content="0">
        <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
        <meta http-equiv="description" content="This is my page">
        <!--
        <link rel="stylesheet" type="text/css" href="styles.css">
         -->
    </head>
    <body>系统登录成功!<br>
    </body>
</html>
```

3. 创建 failure.jsp 文件

failure.jsp 显示用户登录失败的信息，failure.jsp 源代码如下所示。

<!-- 例程 9-4 failure.jsp -->

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<%
String path = request.getContextPath();
String basePath = request.getScheme() + "://" + request.getServerName() + ":" + request.
getServerPort() + path + "/";
 %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <base href="<%=basePath%>">
        <title>登录失败</title>
        <meta http-equiv="pragma" content="no-cache">
        <meta http-equiv="cache-control" content="no-cache">
        <meta http-equiv="expires" content="0">
        <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
        <meta http-equiv="description" content="This is my page">
        <!--
        <link rel="stylesheet" type="text/css" href="styles.css">
         -->
    </head>
    <body>用户名或密码错误,系统登录失败!<br>
    </body>
</html>
```

4. 创建 LoginForm.java 文件

LoginForm.java 用来存放表单数据并进行表单验证，如果用户没有输入用户名和密码就提交表单，将返回出错信息，其源代码如下。

<!-- 例程 9-5 LoginForm.java -->

```
package com.my.struts.form;
import javax.servlet.http.HttpServletRequest;
```

```
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;
public class LoginForm extends ActionForm {
    /** 密码 */
    private String password;
    /** 用户名 */
    private String username;
    /**
     * validate 方法,用于检验用户名和密码是不是为空
     */
    public ActionErrors validate(ActionMapping mapping,
            HttpServletRequest request) {
        // TODO Auto-generated method stub
        ActionErrors errors = new ActionErrors();
        if ((username == null)|| (username.length()<1)){
            errors.add("username",new ActionMessage("login.no.username.erros"));
        }
        if ((password == null)|| (password.length()<1)){
            errors.add("password",new ActionMessage("login.no.password.erros"));
        }
        return errors;
    }
    /**
     * reset 方法
     */
    public void reset(ActionMapping mapping, HttpServletRequest request) {
        this.username = null;
        this.password = null;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
}
```

【说明】

当用户提交了登录页面表单后,Struts 框架将自动把表单数据组装到 ActionForm Bean 中。接下来,Struts 框架会自动调用 ActionForm Bean 的 validate()方法进行表单验证。如果 validate()方法返回的 ActionErrors 对象为 null,或者不包含任何 ActionMessage 对象,就表示没有错误,数据验证通过。如果 ActionErrors 中包含 ActionMessage 对象,就表示发生了验证错误,Struts 框架会把 ActionErrors 对象保存到 request 范围内,然后把请

求转发到某个 JSP 页面,通过<html:errors>标签把 request 范围内的 ActionErrors 对象中包含的错误消息显示出来,提示用户名没有输入。代码 errors. add("username",new ActionMessage("login. no. username. erros"))就是当用户名(username)没有输入时,将一个错误信息加到 errors 中,这里,login. no. username. erros 是一个消息文本的名字,在资源文件 ApplicationResources. properties 中进行定义。该资源文件内容如图 9-6 所示。

name	value
login.no.username.erros	请输入用户名!
login.no.password.erros	请输入密码!

图 9-6 资源文件 ApplicationResources. properties 内容

9.3.4 创建控制器组件

控制器包括一个 Action 类 LoginAction,它进行业务逻辑验证,如果用户输入的用户名或密码错误,将返回错误消息,并决定将合适的视图组件返回给用户。LoginAction. java 源代码如下所示。

<!-- 例程 9 - 6 LoginAction. java -->

```
package com.my.struts.action;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import com.my.struts.form.LoginForm;
/**
 * @struts.action path = "/login" name = "loginForm" input = "/login.jsp" scope = "request"
validate = "true"
  */
public class LoginAction extends Action {
    /**
     * execute 方法,用于执行检验用户名和密码是否正确
     */
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        LoginForm loginForm = (LoginForm) form;
        String username = loginForm.getUsername();
        String password = loginForm.getPassword();
        if ((username.equals("admin")) && (password.equals("admin")))
            return mapping.findForward("success");
        else
            return mapping.findForward("failure");
    }
}
```

【说明】

(1) 程序中通过 LoginAction 的 execute()方法从 LoginForm Bean 获得数据。

(2) if ((username. equals("admin")) && (password. equals("admin")))的作用是 LoginAction 把控制转给合适的视图组件,以显示提示信息。在成功登录的情况下,将控制转给 success。其中 success 和 failure 是一个请求转发路径的名字,在 Struts 的配置文件 struts-config. xml 中定义。

9.3.5 创建配置文件

1. 配置 Struts 框架

Struts 框架的配置文件 struts-config. xml 是一个非常重要的文件,在启动时会读入其配置文件,根据它来创建和配置各种 Struts 组件。struts-config. xml 源代码如下。

```
<?xml version = "1.0" encoding = "UTF - 8"?>
<! DOCTYPE struts - config PUBLIC " - //Apache Software Foundation//DTD Struts Configuration
1.2//EN" "http://struts.apache.org/dtds/struts - config_1_2.dtd">
<struts - config>
  <data - sources />
  <form - beans >
    <form - bean name = "loginForm" type = "com.my.struts.form.LoginForm" />
  </form - beans>
  <global - exceptions />
  <global - forwards >
    <forward name = "success" path = "/success.jsp" />
    <forward name = "failure" path = "/failure.jsp" />
  </global - forwards>
  <action - mappings >
    <action
      attribute = "loginForm"
      input = "/login.jsp"
      name = "loginForm"
      path = "/login"
      scope = "request"
      type = "com.my.struts.action.LoginAction" />
  </action - mappings>
  <message - resources parameter = "com.my.struts.ApplicationResources" />
</struts - config>
```

【说明】

(1) <form-beans>元素配置了与 HTTP 请求参数中的名称相对应,name 属性指定 Form Bean 为 loginForm,type 属性指定 FormBean 的完整类名,控制器将请求参数传递到 ActionForm Bean 的实例,然后将这个实例传送到 Action 类。

(2)
```
<forward name = "success" path = "/success.jsp" />
<forward name = "failure" path = "/failure.jsp" />
```

<forwards>元素配置了与 sendRedirect()方法转向的页面,name 属性指定转向的名称为 success,path 属性指定转向页面的路径为/success. jsp。

```
(3) <action-mappings>
       <action
        attribute="loginForm"
        input="/login.jsp"
        name="loginForm"
        path="/login"
        scope="request"
        type="com.my.struts.action.LoginAction" />
    </action-mappings>
```

struts-config.xml 通过<action>元素配置了一个 Action 组件，其中<action>元素的 path 属性指定请求访问 Action 的路径为/login，type 属性指定 Action 的完整类名，name 属性指定需要传递给 Action 的 ActionForm Bean 为 loginForm，scope 属性指定 ActionForm Bean 的存放范围为 request，input 属性指定当表单验证失败时的转发路径为/login.jsp。

```
(4) <message-resources parameter="com.my.struts.ApplicationResources" />
```

<message-resources>元素指定了资源文件 properties 的参数名为 com.my.struts.Application-Resources。

上述一些 struts-config.xml 文件配置关系如图 9-7 所示。

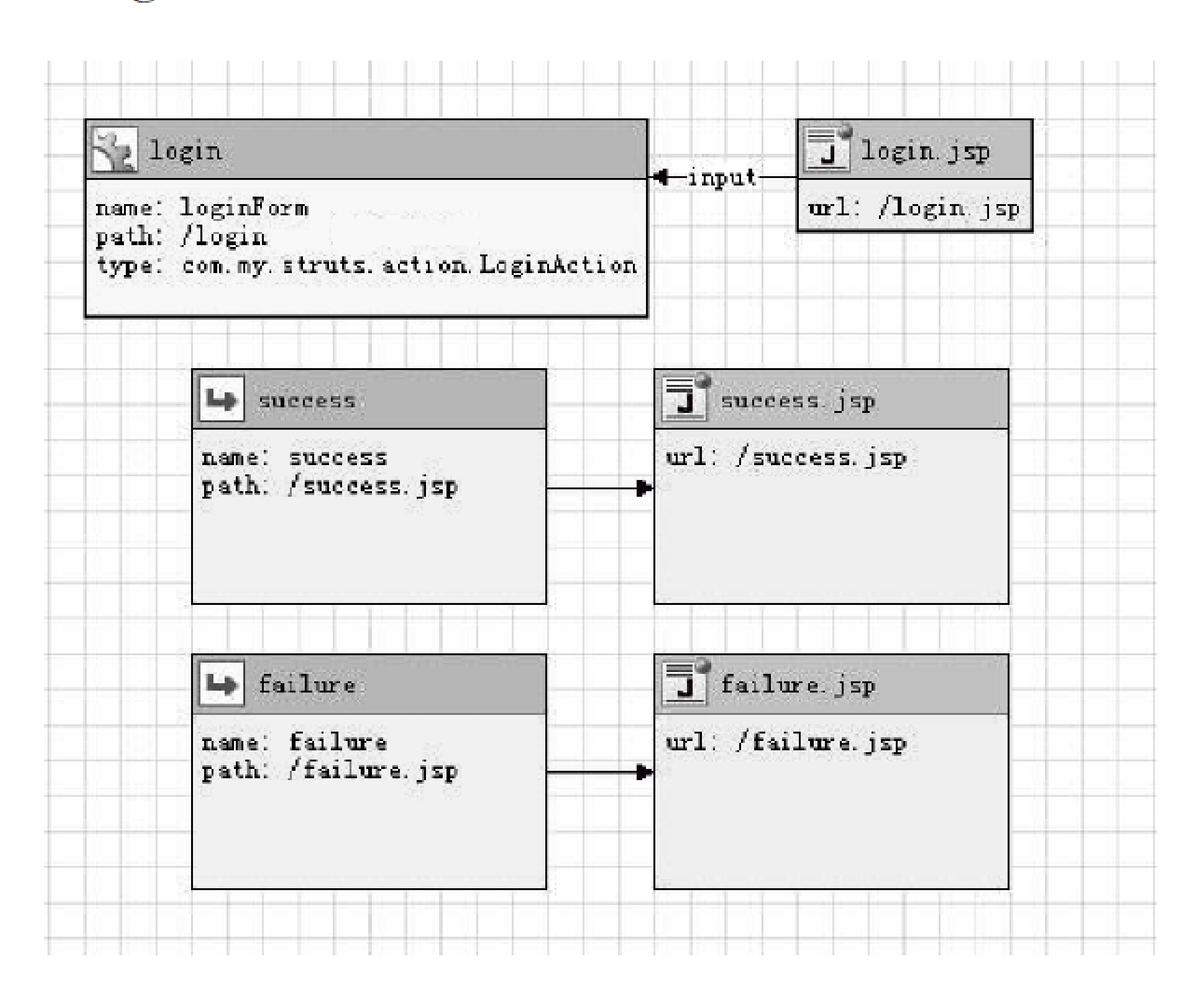


图 9-7　struts-config.xml 文件配置关系

2. 配置 Web 应用

对于 Struts 应用，它的配置文件 Web 应该对 ActionServlet 类进行配置。此外，还应该声明 Web 应用所使用到的 Struts 标签库。web.xml 代码如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
 <servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
  <init-param>
```

```
    <param-name>config</param-name>
    <param-value>/WEB-INF/struts-config.xml</param-value>
   </init-param>
   <init-param>
    <param-name>debug</param-name>
    <param-value>3</param-value>
   </init-param>
   <init-param>
    <param-name>detail</param-name>
    <param-value>3</param-value>
   </init-param>
   <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet-mapping>
   <servlet-name>action</servlet-name>
   <url-pattern>*.do</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
   <welcome-file>login.jsp</welcome-file>
  </welcome-file-list>
  <login-config>
   <auth-method>BASIC</auth-method>
  </login-config>
</web-app>
```

【说明】

(1) web.xml 通过<servlet>元素配置了 ActionServlet 的属性。

(2) 通过<servlet-mapping>元素配置了 URL 的匹配形式，表明 ActionServlet 负责处理所有以.do 为后缀的 URL。如在 login.jsp 中就用<html:form action="/login.do">或<html:form action="/login">的形式提交请求给 ActionServlet。

(3) 通过<welcome-file-list>元素配置了 Web 程序的默认首页为 login.jsp 页面。

9.3.6 部署和运行 Struts 程序

1. 添加 Struts 软件包

在编写程序之前，还需要把 Struts 框架所需要的 JAR 文件和标签库描述文件 TLD 部署到 Tomcat 中。

(1) Struts 的 JAR 文件如图 9-8 所示。

(2) Struts 的 TLD 文件如图 9-9 所示。

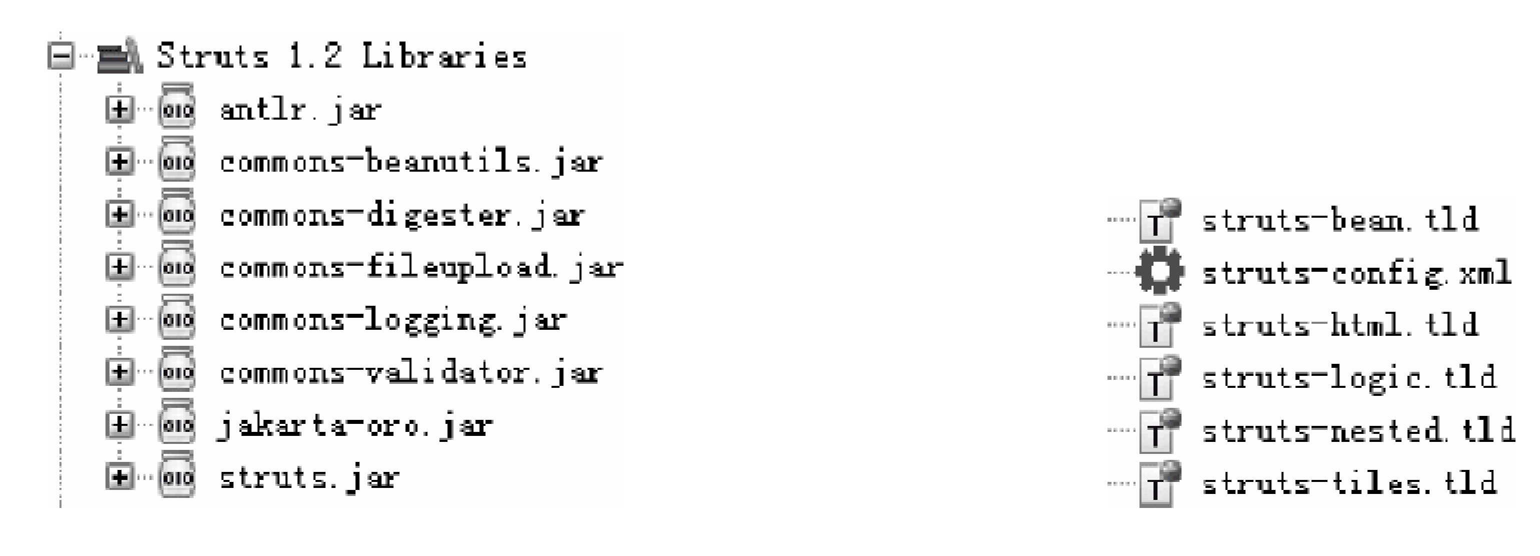


图 9-8 Struts 的 JAR 文件　　　图 9-9 Struts 的 TLD 文件

(3) 登录程序的目录结构如图 9-10 所示。

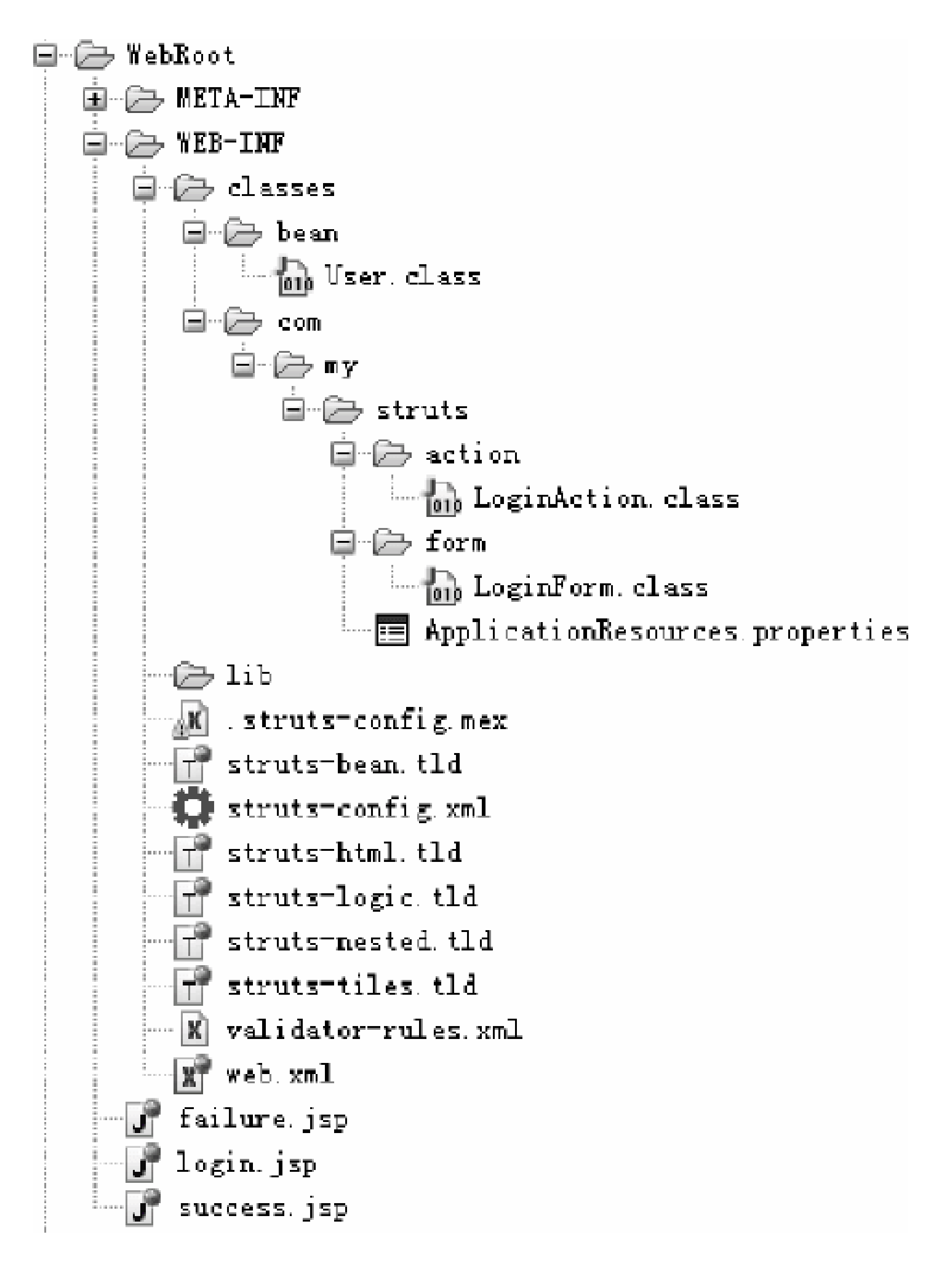


图 9-10　登录程序的目录结构

2. 运行程序

该 Web 程序运行后的初始界面如图 9-11 所示。

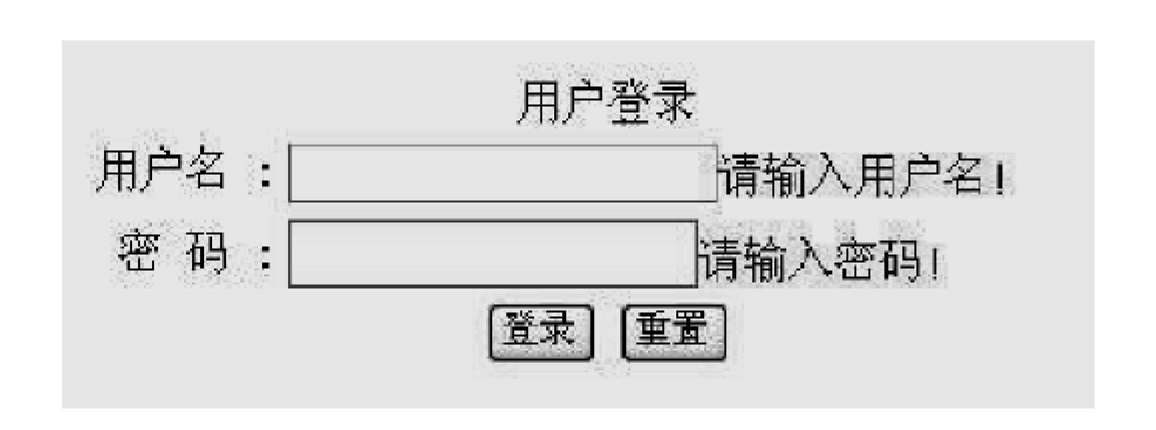


图 9-11　用户登录未输入用户名和密码错误提示

在登录页面中要求必须输入用户名和密码，如果没有输入这些信息，则表单验证不能通过，系统提示必须输入。输入正确的用户名和密码后显示“系统登录成功！”界面；若是输入错误则显示“用户名或密码错误，系统登录失败！”界面。

9.4　上机指导

9.4.1　数据库登录程序设计

1. 练习目标

(1) 熟练掌握 Struts 框架结构。

(2) 熟练掌握 Struts 框架 Web 程序的设计。

（3）熟练掌握数据库访问程序的设计。

（4）熟练掌握 Struts 框架程序的部署和运行。

2. 练习指导

添加一个数据库 jsp_db，数据库中有一个 users＜用户＞数据库表，该表中存放着用户的信息，包括用户名＜username＞和密码＜password＞等。

（1）users 表结构和数据如图 9-12 和图 9-13 所示。

列名	数据类型	长度	允许空
id	int	4	
username	varchar	20	✓
password	varchar	20	✓

图 9-12 users 表结构

id	username	password
1	admin	admin
2	chen	chen

图 9-13 users 表数据信息

（2）创建模型组件——连接数据库 DBConnect.java，源代码如下。

```
<!-- 例程 9-7 DBConnect.java -->

package util;
import java.sql.*;
public class DBConnect {
    public DBConnect() {
    }
    public Connection getConnection(){
        Connection con = null;

        //JDBC 驱动程序名称
        String drivername = "com.microsoft.jdbc.sqlserver.SQLServerDriver";

        //连接数据库 url
        String url = "jdbc:microsoft:sqlserver://localhost:1433;DatabaseName = jsp_db";

        //连接数据库用户名为 sa
        String username = "sa";

        //连接数据库密码为空
        String password = "";
        try{
            //加载 JDBC 驱动程序
            Class.forName(drivername);
            //连接数据库
            con = DriverManager.getConnection(url,username,password);
            System.out.println("数据库连接成功!");
        }catch(ClassNotFoundException e){
            e.printStackTrace();
        }catch(SQLException e){
            e.printStackTrace();
        }
```

```
            return con;
        }
    }
```

(3) 创建模型组件——访问数据库表 users 信息的 UserUtil.java,源代码如下。

<!-- 例程 9-8 UserUtil.java -->

```
package util;
import java.sql.*;
public class UserUtil {
    private Connection con;

    //查询数据库表中的信息是否存在
    public boolean findUser(String username,String password){
        con = (new DBConnect()).getConnection();
        boolean flag = false;
        Statement stmt;
        ResultSet rs;
        String sql = "select * from users where username = '" + username + "' and password = '" +
password + "'";
        try{
            stmt = con.createStatement();
            rs = stmt.executeQuery(sql);
            if (rs.next())
                flag = true;
            rs.close();
            stmt.close();
            con.close();
        }catch(Exception e){
            e.printStackTrace();
        }
        return flag;
    }
}
```

(4) 修改控制器组件——LoginAction.java,源代码如下。

<!-- 例程 9-9 LoginAction.java -->

```
package com.my.struts.action;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import com.my.struts.form.LoginForm;
import util.UserUtil;
/**
 * @struts.action path = "/login" name = "loginForm" input = "/login.jsp" scope = "request"
validate = "true"
```

```
 */
public class LoginAction extends Action {
    /**
     * Method execute
     * @return ActionForward
     */
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        LoginForm loginForm = (LoginForm) form;// TODO Auto-generated method stub
        String username = loginForm.getUsername();
        String password = loginForm.getPassword();
        UserUtil userutil = new UserUtil();
        if (userutil.findUser(username, password))
            return mapping.findForward("success");
        else
            return mapping.findForward("failure");
    }
}
```

部署和运行该程序。

9.4.2 注册用户信息

1. 练习目标

(1) 熟练掌握 MVC 程序设计的方法。

(2) 熟练掌握 Struts 框架 Web 程序的设计。

(3) 熟练掌握数据库程序存取的设计。

(4) 熟练掌握 Struts 框架程序的部署和运行。

2. 练习指导

在上个实验的基础上完成对用户信息的注册,即在 users<用户>表中添加用户的信息,包括用户名<username>和密码<password>等。

(1) 修改模型组件——UserUtil.java,源代码如下。

<!-- 例程 9-10 UserUtil.java -->

```
package util;
import java.sql.*;
import bean.User;
public class UserUtil {
    private Connection con;

    //查询数据库表中的信息是否存在
    public boolean findUser(String username,String password){
        con = (new DBConnect()).getConnection();
        boolean flag = false;
        Statement stmt;
```

```
        ResultSet rs;
        String sql = "select * from users where username = '" + username + "'and password = '" +
password + "'";
        try{
            stmt = con.createStatement();
            rs = stmt.executeQuery(sql);
            if (rs.next())
                flag = true;
            rs.close();
            stmt.close();
            con.close();
        }catch(Exception e){
            e.printStackTrace();
        }
        return flag;
    }

    //向数据库表中添加一个 user 的信息
    public boolean addUser(User user){
        con = (new DBConnect()).getConnection();
        PreparedStatement pstmt;
        String sql = "insert into users values(?,?)";
        boolean flag = false;
        try{
            pstmt = con.prepareStatement(sql);
            pstmt.setString(1, user.getUserName());
            pstmt.setString(2, user.getPassword());
            pstmt.execute();
            flag = true;
            pstmt.close();
            con.close();
        }catch(Exception e){
            e.printStackTrace();
        }
        return flag;
    }
```

(2) 创建视图组件,其相关程序的源代码如下。

```
<!-- 例程 9-11 addUser.jsp -->

<%@ page language = "java" pageEncoding = "gb2312" %>
<%@ taglib url = "http://struts.apache.org/tags-bean" prefix = "bean" %>
<%@ taglib url = "http://struts.apache.org/tags-html" prefix = "html" %>
<html>
    <head>
        <title>添加用户信息</title>
    </head>
    <body>
        <html:form action = "/addUser">
        <table align = "center" border = "0" width = "400">
```

```
            <tr>
            <td colspan = "2" align = "center">用户注册</td>
            </tr>
            <tr>
            <td width = "80" align = "right">用户名:</td>
            <td><html:text property = "username"/><html:errors property = "username"/></td>
            </tr>
            <tr>
            <td align = "right">设置密码: </td>
            <td><html:password property = "password"/><html:errors property = "password"/></td>
            </tr>
            <tr>
            <td align = "right">验证密码: </td>
            <td><html:password property = "repassword"/> < html: errors property =
"repassword"/></td>
            </tr>
            <tr>
             <td colspan = "2" align = "center"><html:submit value = "登录"/>  <html:
reset value = "重置"></html:reset></td>
            </tr>
        </table>
        </html:form>
    </body>
</html>
```

<!-- 例程 9-12 AddUserForm.jsp -->

```
package com.my.struts.form;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;
/**
 * @struts.form name = "addUserForm"
 */
public class AddUserForm extends ActionForm {
    /*
     * Generated fields
     */

    /** password property */
    private String password;

    /** username property */
    private String username;

    /** repassword property */
    private String repassword;

    /**
     * Method validate
```

```
 * @return ActionErrors
 */
public ActionErrors validate(ActionMapping mapping,
        HttpServletRequest request) {
    // TODO Auto-generated method stub
    ActionErrors errors = new ActionErrors();

    if ((username == null)|| (username.length()<1)){
        errors.add("username",new ActionMessage("reg.no.username.errors"));
    }
    if ((password == null)|| (password.length()<1)){
        errors.add("password",new ActionMessage("reg.no.password.errors"));
    }
    if ((password.equals(repassword))){
        errors.add("repassword",new ActionMessage("reg.notsame.password.errors"));
    }
    return errors;
}

/**
 * Method reset
 * @param mapping
 * @param request
 */
public void reset(ActionMapping mapping, HttpServletRequest request) {
    // TODO Auto-generated method stub
}

/**
 * Returns the password.
 * @return String
 */
public String getPassword() {
    return password;
}

/**
 * Set the password.
 * @param password The password to set
 */
public void setPassword(String password) {
    this.password = password;
}

/**
 * Returns the username.
 * @return String
 */
public String getUsername() {
    return username;
}
```

```
    /**
     * Set the username.
     * @param username The username to set
     */
    public void setUsername(String username) {
        this.username = username;
    }

    /**
     * Returns the repassword.
     * @return String
     */
    public String getRepassword() {
        return repassword;
    }

    /**
     * Set the repassword.
     * @param repassword The repassword to set
     */
    public void setRepassword(String repassword) {
        this.repassword = repassword;
    }
}
```

(3) 创建控制器组件 AddUserAction.java,其源代码如下。

<!-- 例程 9-13 addUserAction.java -->

```
package com.my.struts.action;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import util.UserUtil;
import bean.User;
import com.my.struts.form.AddUserForm;

/**
 * @struts.action path = "/addUser" name = "addUserForm" input = "/addUser.jsp" scope =
"request" validate = "true"
 */
public class AddUserAction extends Action {

    /**
     * Method execute
     * @return ActionForward
     */
```

```
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        AddUserForm addUserForm = (AddUserForm) form;// TODO Auto-generated method stub
        String username = addUserForm.getUsername();
        String password = addUserForm.getPassword();
        User user = new User();
        user.setUserName(username);
        user.setPassword(password);
        UserUtil userutil = new UserUtil();
        if (userutil.addUser(user))
            return mapping.findForward("addUsersuccess");
        else
            return mapping.findForward("addUserfailure");
    }
}
```

（4）修改配置文件 struts-config.xml，其源代码如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
1.2//EN" "http://struts.apache.org/dtds/struts-config_1_2.dtd">

<struts-config>
  <data-sources />
  <form-beans >
    <form-bean name="loginForm" type="com.my.struts.form.LoginForm" />
    <form-bean name="addUserForm" type="com.my.struts.form.AddUserForm" />
  </form-beans>

<global-exceptions />
   <global-forwards >
    <forward name="success" path="/success.jsp" />
    <forward name="failure" path="/failure.jsp" />
   </global-forwards>

  <action-mappings >
    <action
      attribute="loginForm"
      input="/login.jsp"
      name="loginForm"
      path="/login"
      scope="request"
      type="com.my.struts.action.LoginAction" />
    <action
      attribute="addUserForm"
      input="/addUser.jsp"
      name="addUserForm"
      path="/addUser"
      scope="request"
      type="com.my.struts.action.AddUserAction" />
  </action-mappings>
```

```
  <message-resources parameter="com.my.struts.ApplicationResources" />
</struts-config>
```

其他一些页面的创建与设置请读者自行设计。

本章小结

Struts 是一种基于 MVC 设计模式的 Java Web 框架，它使系统开发过程各个模块更加细化。利用 taglib 获得可重用的代码；利用 ActionServlet 配合 struts-config.xml 实现对整个系统导航，增强了开发人员对系统的整体把握；用户界面、业务逻辑和业务控制的分离，使系统的层次结构更加清晰，易于分工协作，同时增强系统的可扩展性和维护性。

习题 9

一、简答题

1. 简述 Struts 框架的基本结构及工作流程。
2. 简述 Struts 框架中控制器的特征，以及如何在配置文件 struts.xml 文件中配置。
3. 简述如何实现 JSP 页面国际化及校验错误信息国际化。

二、操作编程题

根据 9.4.1 小节和 9.4.2 小节完成对用户注册、修改、删除和密码修改功能的 struts 框架程序。

第10章 上机指导综合范例

前 9 章分别介绍了 JSP 概念、运行环境、页面组成、各种技术的应用方法。本章将综合各章概念、技术和方法，把这些概念、技术和方法运用到实际应用当中，使读者对 JSP 技术和应用有更深刻的理解。

本章主要内容：

- 设计原理和设计方法的应用；
- 设计合理的程序结构；
- 综合运用各项编程技术和方法。

10.1 成绩管理系统

成绩管理系统对学校加强学生成绩管理有着极其重要的作用，建立成绩管理系统，采用计算机对学生成绩进行管理，进一步提高办学效益和现代化水平，帮助广大教师提高工作效率，实现学生成绩信息管理工作流程的系统化、规范化和自动化。本节介绍如何开发一个简单的成绩管理系统，本系统实现用户登录、成绩管理、成绩录入、修改、查询、删除 6 个模块。

10.1.1 设计原理

1. 系统构成

本系统由 6 个模块组成，分别是用户登录模块、成绩管理模块、成绩录入模块、成绩修改模块、成绩查询模块、成绩删除模块。其模块结构如图 10-1 所示。

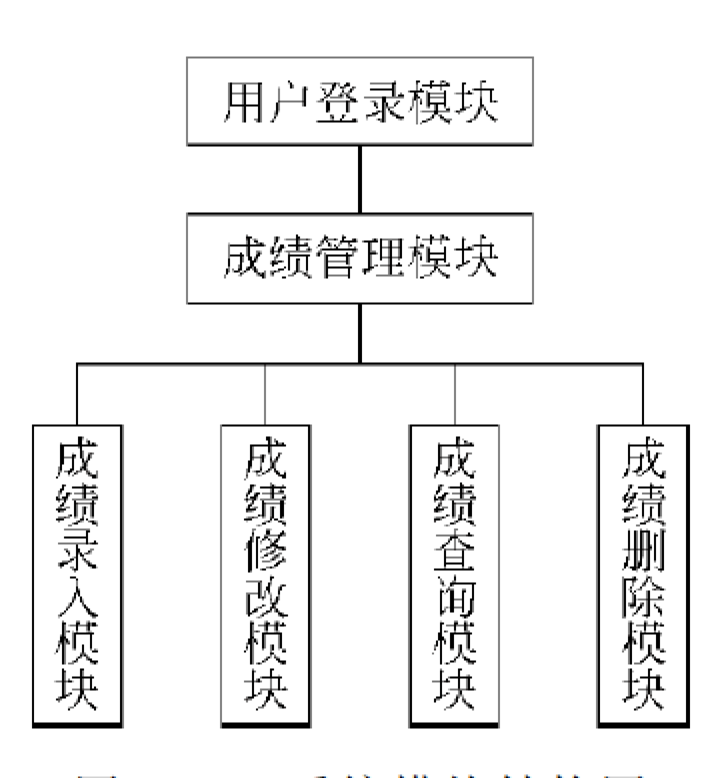


图 10-1 系统模块结构图

各模块功能如下。

- 用户登录模块：用于登录客户输入用户名和密码，经系统验证后转入成绩管理页面。
- 成绩管理模块：登录成功后显示该模块，负责与其他页面进行交互，通过该模块进入成绩录入、成绩修改、成绩查询及成绩删除界面。
- 成绩录入模块：用于录入学生的成绩，并将成绩保存到数据表中。
- 成绩修改模块：用户可以通过此模块来修改学生的成绩。
- 成绩查询模块：通过此模块用户可以根据学号查询学生的成绩。

- 成绩删除模块：通过此模块用户可以根据学号删除学生的成绩。

2. 数据库设计

本系统使用的是 Access 数据库，数据库为新建立的 cjgl.mdb，该数据库中包含两张表，存放系统中所需要的数据。

（1）学生成绩表(students)：该表用于保存学生的成绩，其表结构如表 10-1 所示。

表 10-1　学生表结构

字段名	数据类型	字段宽度
number	文本	5
name	文本	10
math	单精度数字	自动
english	单精度数字	自动
phics	单精度数字	自动

（2）用户登录网站的账号表(Login)：用于保存用户账号，其表结构如表 10-2 所示。

表 10-2　账号表的结构

字段名	数据类型	字段宽度
XM	文本	10
PW	文本	10

10.1.2　用户登录

在网站设计中，希望某些网页只有具有特定权限的用户才能访问需要建立登录机制，由系统管理员给访问者分配账号，只有具有账号的客户才能访问网页。本系统实现登录模块的页面由 LogFrm.htm 和 Login.jsp 组成。

LogFrm.htm 页面检查输入账号的合法性，Login.jsp 检查输入账号的正确性。其页面关系如图 10-2 所示。

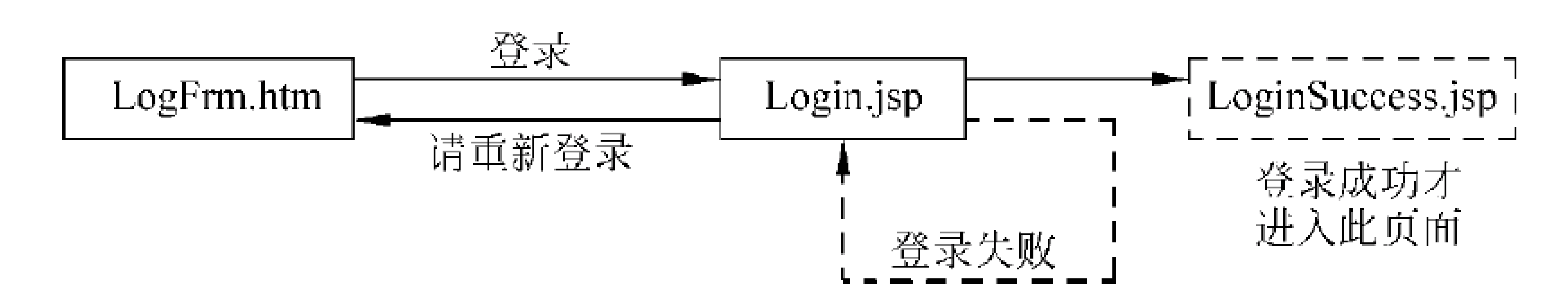


图 10-2　登录页面交互图

在 LogFrm.htm 页面中创建一个表单，包含两个文本框，用于登录客户输入用户名和密码。该页面的源代码如下。

```
<! -- 例程 10 - 1 LogFrm.htm -- >

<HTML>
  <HEAD>
    <TITLE>用户登录</TITLE>
```

```
        <SCRIPT Language = javascript>
        <!--
         //下面是对用户账号和密码检查的函数定义
           function datacheck()
           {
             //下面的 if 判断语句将检查是否输入账号
             if(frmLogin.UserNm.value == "")
             {
               window.alert("您必须完成账号的输入!");     //显示错误信息
               document.frmLogin.elements(0).focus();    //将光标移至账号输入栏
               return ;
             }
              //下面的 if 判断语句将检查是否输入密码
             if(frmLogin.UserPasswd.value == "")
             {
               window.alert("您必须完成密码的输入!");
               document.frmLogin.elements(1).focus();    //将光标移至密码输入栏
               return ;
             }
             frmLogin.submit();       //送出表单中的资料
           }
           -->
         </SCRIPT>
        </HEAD>
      <BODY>
        <CENTER><FONT SIZE = 5 COLOR = blue><b>用户登录</b></FONT>
        <HR>
        <FORM action = "Login.jsp" method = "post" name = "frmLogin">
          <FONT color = midnightblue size = 4><STRONG>
          用户名称: <INPUT name = "UserNm" ><P></P>
          用户密码: <INPUT name = "UserPasswd" type = password>
          <P></P>
          </STRONG></FONT>
          <!-- 按下此指令按钮可进行资料检查并送出表单资料 -->
          <INPUT type = "button" value = "登 录" onclick = "datacheck()">
        </FORM>
      </CENTER>
    </BODY>
  </HTML>
```

【说明】

在该段代码中定义了一个函数 datacheck(),用于检查账号和密码输入的完整性。Login.jsp 页面的源代码如下。

```
                    <!-- 例程 10-2 Login.jsp -->

<%@ page contentType = "text/html; charset = GB2312" import = "java.sql.* " %>
<%@ page import = "java.io.* " %>
  <%!
    String CheckLogin(String Login1, String Password1) throws Exception
    {
```

```
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;
        String result = null;
        String Login = Login1.trim();
        String Password = Password1.trim();
        try
        {
          Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
          con = DriverManager.getConnection("jdbc:odbc:grade");
          stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                 ResultSet.CONCUR_READ_ONLY);
          String strSQL = "SELECT * FROM Login " + "WHERE XM = '" + Login + "'";
          rs = stmt.executeQuery(strSQL);  //执行 SQL 语句,进行账号查询
          if(!rs.next())                   //检查游标是否指到最后一条记录
             result = "无此账号";           //若指向最后一条记录则表示没有记录
          else
            if(!rs.getString("PW").equals(Password))     //判断密码是否正确
               result = "密码错误";
            else
               result = "成功登录";
       }
      catch(Exception ex)
      {
         throw ex;
       }
      finally
      {
          rs.close();
          stmt.close();
          con.close();
       }
      return result;
    }
  %>
 <%
   String UserNm = request.getParameter("UserNm");
                                          //取得表单输入的账号
   String UserPasswd  = request.getParameter("UserPasswd");
                                          //取得表单输入的密码
   if(UserNm == null || UserPasswd == null)
       response.sendRedirect("LogFrm.htm");
   String strCheckLogin = CheckLogin(UserNm, UserPasswd);  //进行账号与密码的检查
   if (strCheckLogin.equals("成功登录"))                    //判断是否成功登录
   {
        session.setAttribute("UserNm",UserNm);
                                    //将登录账号储存在 session 中
        session.setAttribute("UserPasswd",UserPasswd);
                                    //将账号密码储存在 session 中
        response.sendRedirect("LoginSuccess.jsp");
                                    //将显示网页导向 LoginSuccess.jsp 网页
```

```
    }
  %>

<!--若登录失败则会执行下面的语句-->
<HTML>
  <HEAD>
    <TITLE>用户登录</TITLE>
  </HEAD>
  <BODY>
    <CENTER>
      <FONT SIZE = 5 COLOR = blue>使用者登录</FONT>
  </CENTER>
  <HR>
  <Center>
      <% = strCheckLogin %>        //输出登录失败原因
      <P></P>
      <A href = "LogFrm.htm">请重新登录</A>
   </Center>
  </BODY>
</HTML>
```

【说明】

该段代码中定义了函数 CheckLogin(String Login1, String Password1),用于检查账号和密码的正确性。若账号或密码为空,则转向 LogFrm.htm 页面;若账号和密码正确,则将账号和密码保存到 session 中,并转向成绩管理页面(LoginSuccess.jsp);若登录失败,则输出登录失败的原因并建立超链接,连接到 LogFrm.htm 页面。

10.1.3 成绩管理

该模块由 LoginSuccess.jsp 页面实现,通过该界面进入成绩录入、成绩修改、成绩查询、成绩删除界面。该模块与其他页面的交互关系如图 10-3 所示。

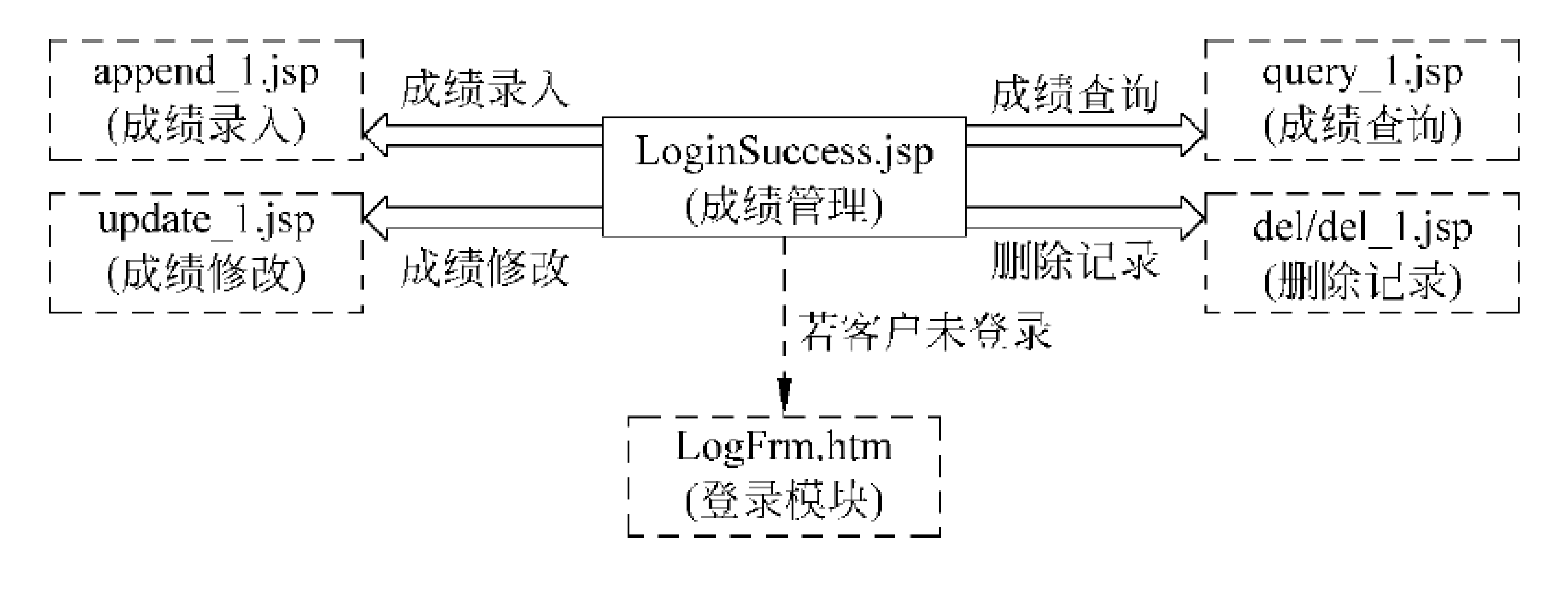


图 10-3　成绩管理模块交互图

LoginSuccess.jsp 程序源代码如下。

<!--例程 10-3 LoginSuccess.jsp-->

```
<%@ page contentType = "text/html; charset = GB2312" %>
 <%
      //检查用户是否已经完成登录
```

```
        String Name = (String)session.getAttribute("UserNm");
        if(Name == null)          //若 Name 变量为 null 代表尚未完成登录
          {
            response.sendRedirect("LogFrm.htm");
        }
    %>
<HTML>
  <HEAD>
    <TITLE>成绩管理</TITLE>
  </HEAD>
  <BODY>
    <CENTER>
        <FONT SIZE = 5 COLOR = blue> <b>成绩管理</b></FONT>
        <HR>
        <h2> <font size = 3 color = yellow>管理员:<% = Name %> </font> </h2>
        <P></P>
        <Table>
        <TR align = left>
            <TD width = 78>
                        <a href = "append/append_1.jsp">成绩录入</A>
            </TD>
            <TD width = 78>
                        <a href = "update/update_1.jsp">成绩修改</A>
            </TD>
            <TD width = 78>
                       <a href = "query/query_1.jsp">成绩查询</A>
            </TD>
            <TD width = 78>
                       <a href = "del/del_1.jsp">删除记录</A>
            </TD>
        </TR>
      </Table><BR>
    </CENTER>
  </BODY>
</HTML>
```

【说明】

该段代码中获取 session 中的账号 Name,若账号为空(表示客户还未登录),则定向到 LogFrm.htm 页面。该页面中创建了 4 个超链接,分别连接到其他 4 个页面。

10.1.4 成绩录入

该模块提供一个界面,用户在此界面中录入学生成绩。

该模块由两个页面组成,append_1.jsp 页面提供成绩录入界面,把成绩提交给 append_2.jsp 页面,由 append_2.jsp 页面把成绩保存到数据表(students)中。其页面交互关系如图 10-4 所示。

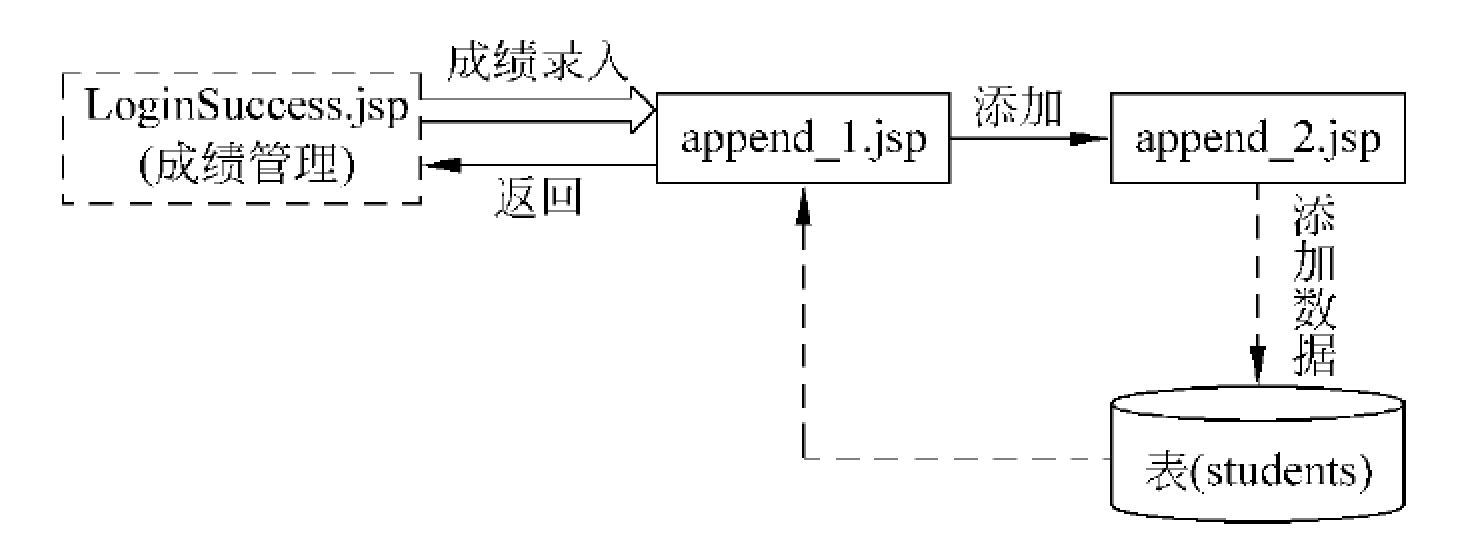


图 10-4 成绩录入模块交互图

append_1.jsp 程序源代码如下。

<! -- 例程 10 - 4 append_1.jsp -- >

```
<%@ page contentType = "text/html;charset = GB2312" %>
<%@ page import = "java.sql.*" %>
<HTML>
<HEAD>
<TITLE>成绩录入</TITLE>
<body>
<center>
    <p><font size = 5><b> 成绩录入 </b></font>
    <FONT  size = 4>
    <FORM  action = "append_2.jsp" method = post>
      同学学号:<Input type = "text" name = "number"><BR>
      同学姓名:<Input type = "text" name = "name">  <BR>
      数学成绩:<Input type = "text" name = "math">  <BR>
      英语成绩:<Input type = "text" name = "english"><BR>
      物理成绩:<Input type = "text" name = "physics"><BR>
     <Input type = "submit" name = "b" value = "添加">
    </FORM>
    <BR>
    <%
        String  lr = (String)session.getAttribute("tianjia");
        if(lr == null) lr = "";
     %>
   <p><font size = 4 color = red>  数据录入:<% = lr %></font>
   </FONT>
    <br><br>
   <a href  = "../LoginSuccess.jsp">返回</A>
</center>
</BODY>
</HEAD>
</HTML>
```

【说明】

该程序创建一个表单,表单中包含 5 个文本框,这些文本框用于输入学生成绩。提交该表单后,从 session 中获取添加执行标志(tianjia=成功|失败)。同时表单中还有一个超链接,指向 LoginSuccess.jsp 页面。

append_2.jsp 程序源代码如下。

<!-- 例程 10-5 append_2.jsp -->

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.sql.*" %>
<%@ page import="java.io.*" %>
<HTML>
<BODY bgcolor=pink><FONT size=3>
 <%! boolean insert(String number,String name,String m,String e,String p)
       {
         Connection con=null;
         Statement sql=null;
         ResultSet rs=null;
         int num=0;
         try
            {
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            }
         catch(ClassNotFoundException event)
            { }
         try
         {
           con=DriverManager.getConnection("jdbc:odbc:grade");
           sql=con.createStatement();
           String condition="INSERT INTO students VALUES"+"("+"'"+number+"','"+name+
"',"+m+","+e+","+p+")";
           num=sql.executeUpdate(condition);     //执行添加操作
           con.close();
          }
         catch(SQLException event)
           { }
         if (num>0)
            {
                return true ;
            }
         else
            {
                return  false ;
            }
       }
  %>
 <%
      //获取提交的学号
     String number=request.getParameter("number");
     if(number==null)
       {
         number="";
       }
     byte b[]=number.getBytes("ISO-8859-1");
     number=new String(b);
     //获取提交的姓名
     String name=request.getParameter("name");
```

```
        if(name == null)
          {
            name = "";
          }
        byte c[] = name.getBytes("ISO - 8859 - 1");
        name = new String(c);
        //获取提交的新的数学成绩
        String m = request.getParameter("math");
        if(m == null)
          {
            m = " - 100";
          }
         //获取提交的新的英语成绩
         String e = request.getParameter("english");
         if(e == null)
            {
              e = " - 100";
            }
         //获取提交的新的物理成绩
         String p = request.getParameter("physics");
         if(p == null)
            {
             p = " - 100";
            }
      %>
     <%
       if( insert(number,name,m,e,p))
          session.setAttribute("tianjia","成功");
       else
          session.setAttribute("tianjia","失败");
       response.sendRedirect("append_1.jsp");
      %>
   </FONT>
   </BODY>
   </HTML>
```

【说明】

该程序中定义了 boolean insert()方法，该方法向 students 表中添加记录，从表单中获取要添加的数据后，调用 insert(number，name，m，e，p)方法，将数据添加到 students 表中。若添加成功，则把属性值（"tianjia"，"成功"）加入 session 中，否则把属性值("tianjia"，"失败")加入到 session 中，并返回到 append_1.jsp 页面。

10.1.5 成绩修改

该模块提供一个界面，用户通过此界面根据学号修改学生成绩。

该模块由两个页面完成，update_1.jsp 页面提供一个修改成绩的界面，它把修改后的数据提交给 update_2.jsp 页面，update_2.jsp 页面首先在表中查询该学号是否存在，若存在该学号，则执行查询。页面交互关系如图 10-5 所示。

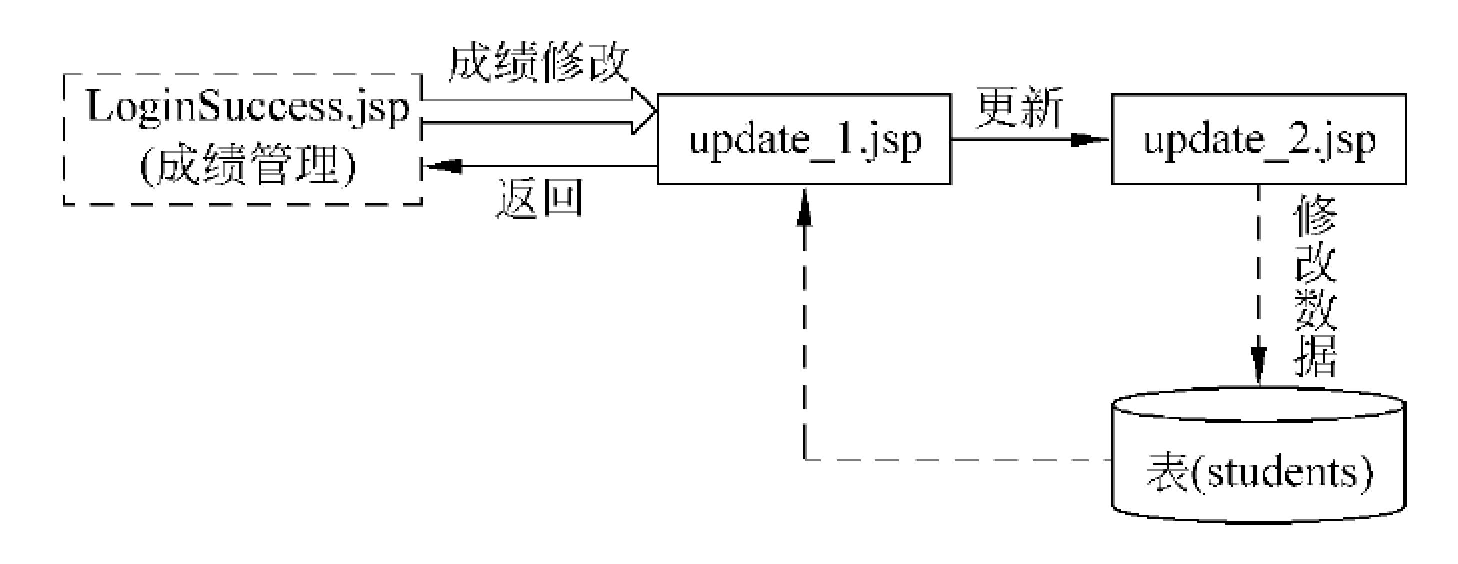


图 10-5 成绩修改模块交互图

update_1.jsp 程序源代码如下。

```
<! -- 例程 10 - 6 update_1.jsp -- >

<% @ page contentType = "text/html;charset = GB2312" %>
<% @ page import = "java.sql. * " %>
<HTML>
<TITLE>成绩修改</TITLE>
<HEAD>
<BODY>
<center>
  <FONT size = 5 > <b> 成绩修改 </b>  </FONT>
  <FONT size = 4 >
   <FORM action = "update_2.jsp" Method = post >
    <br ><br >
      输入修改者的学号: <Input type = "text" name = "number" value = " "> <BR>
      输入新的数学成绩: <Input type = "text" name = "math"  value = 0 >  <BR>
      输入新的英语成绩: <Input type = "text" name = "english" value = 0 > <BR>
      输入新的物理成绩: <Input type = "text" name = "physics" value = 0 > <BR>
      <p></p>< Input type = "submit" name = "b" value = "更新"> <br >
   </FORM>
  </FONT>
  <%
    String  xiugai = (String)session.getAttribute("xiugai");
    out.println(xiugai);
   %>
   <br ><br >
   <a href  =  "../LoginSuccess.jsp">返回</A>
</center>
</BODY>
</HEAD>
</HTML>
```

【说明】

该程序创建一个表单,表单包含 4 个文本框,用于输入关键字(学号)和修改后的成绩。提交该表单后,从 session 中获取修改执行标志(xiugai=成功|失败|无此学号),并输出修改执行标志 xiugai。

update_2.jsp 程序源代码如下。

<! -- 例程 10 - 7 update_2.jsp -- >

```
<%@ page contentType = "text/html;charset = GB2312" %>
<%@ page import = "java.sql.*" %>
<HTML>
<BODY>
<FONT size = 3>
  <%!
    boolean query(String number) //查询是否有学号是 number 的学生
    {
      Connection con = null;
      Statement sql = null;
      ResultSet rs = null;
      try
      {
           Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      }
      catch(ClassNotFoundException e) {}
      try
      {
           con = DriverManager.getConnection("jdbc:odbc:grade");
           sql = con.createStatement();
           String condition = "SELECT * FROM students where number = " + "'" + number + "'" ;
           rs = sql.executeQuery(condition);
           int num = 0;
           while(rs.next()) num++ ;
           con.close();
           if(num > 0)
              return  true;
           else
              return  false ;
       }
       catch(SQLException e)
          { return  false; }
     }
   String update(String number,float newMath,float newEnglish,float newPhysics)
   {
     if(query(number))
       {
         Connection con = null;
         Statement sql = null;
         ResultSet rs = null;
         try
          {
              Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
          }
         catch(ClassNotFoundException e)   {}
         try
          {
              con = DriverManager.getConnection("jdbc:odbc:grade");
              sql = con.createStatement();
```

```
                    String condition1 = "UPDATE students SET math = " + newMath + " WHERE number = " + "'" + number + "'" ;
                    String condition2 = "UPDATE students SET english = " + newEnglish + " WHERE number = " + "'" + number + "'" ;
                    String condition3 = "UPDATE students SET phics = " + newPhysics + " WHERE number = " + "'" + number + "'" ;
                    //执行更新操作
                    sql.executeUpdate(condition1);
                    sql.executeUpdate(condition2);
                    sql.executeUpdate(condition3);
                    con.close();
                    return  "修改成功";
                }
            catch(SQLException e)
                {  return  "修改失败"  ;}
        }
        else
          { return "没有这个学号" ;}
    }
  %>
 <%
    //获取提交的学号
   String number = request.getParameter("number");
   number = number.trim();
   if(number == null)
     {
         number = "";
     }
   byte b[] = number.getBytes("ISO-8859-1");
   number = new String(b);
   //获取提交的新的数学成绩
   String newMath = request.getParameter("math");
   if(newMath == null)
     {
       newMath = "0";
     }
   float  math = Float.parseFloat(newMath);
   //获取提交的新的英语成绩
   String newEnglish = request.getParameter("english");
   if(newEnglish == null)
     {
       newEnglish = "0";
     }
    float  english = Float.parseFloat(newEnglish);
    //获取提交的新的物理成绩
    String newPhysics = request.getParameter("physics");
    if(newPhysics == null)
      {
        newPhysics = "0";
      }
    float  physics = Float.parseFloat(newPhysics);
```

```
    %>
    <%
        String   del = update(number, math,english, physics);
        if(del.equals("修改成功"))
           session.setAttribute("xiugai", "修改成功");
        else
           if(del.equals("修改失败"))
               session.setAttribute("xiugai","修改失败");
           else
               session.setAttribute("xiugai","没有这个学号");
        response.sendRedirect("update_1.jsp");
    %>
</FONT>
</BODY>
</HTML>
```

【说明】

该程序中定义了两个方法，boolean query(String number)方法查询学号为 number 的学生是否存在，若存在返回值为 true，否则返回值为 false。String update()方法修改学号为 number 的成绩，该方法返回值有 3 种情况，分别为“修改成功”、“修改失败”和“没有这个学号”。程序从表单中获取学号和新的成绩数据，然后执行修改操作 del＝update(number, math,english, physics)，根据执行情况，将属性值("xiugai",del)加入到 session 中，然后再重新定向到 update_1.jsp 页面。

10.1.6　成绩查询

该模块根据学号查询学生的成绩，由两个页面组成，在 query_1.jsp 页面中输入学号，然后提交给 query_2.jsp 页面，query_2.jsp 页面完成学生查询。其页面交互关系如图 10-6 所示。

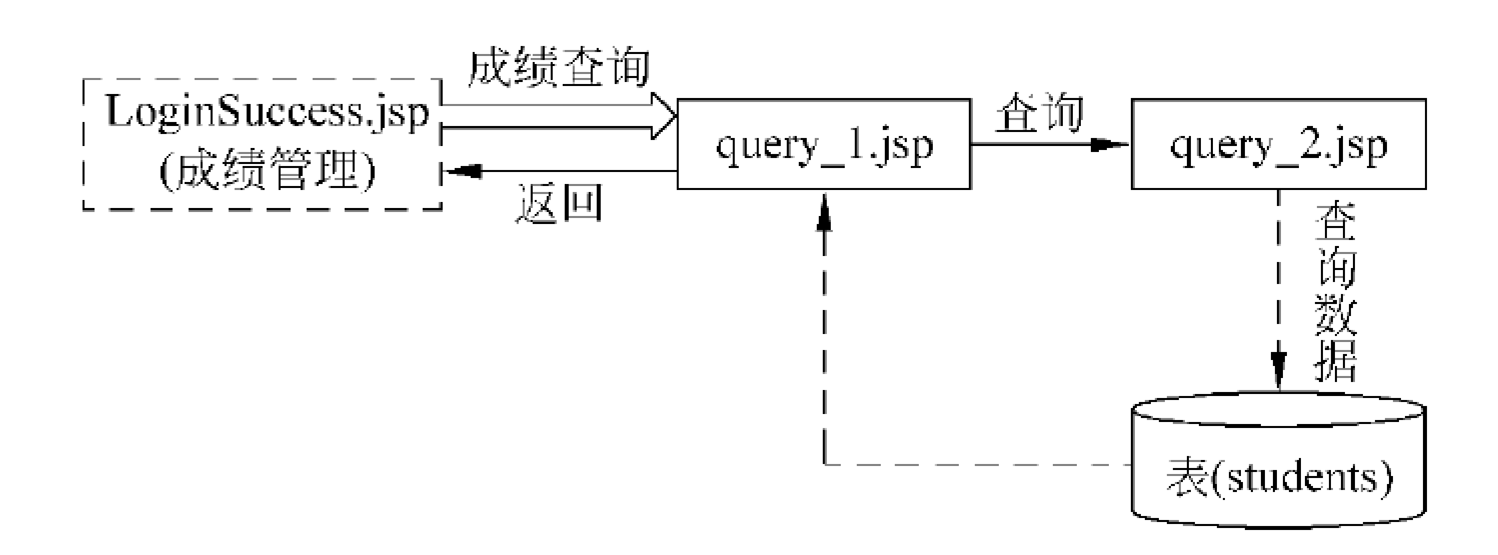


图 10-6　成绩查询模块交互图

query_1.jsp 程序源代码如下。

```
<! -- 例程 10 - 8 query_1.jsp -->

<%@ page contentType = "text/html;charset = GB2312"  %>
<%@ page import  = "java.sql. * "  %>
<HTML>
<HEAD>
<TITLE>成绩查询</TITLE>
```

```
<BODY>
<center>
    <P><font size=5> 成绩查询  </font>
    <FONT size=4>
    <P>
    <FORM action="query_2.jsp" method=post name=form>
       按学号查询:
       <INPUT type="text" name="number" value="">
       <Input type=submit name="g" value="查询">
    </FORM>  <br>
    <%
       String  number=null;
       String  name=null;
       float  math,english,physics;
       Connection con=(Connection)session.getAttribute("con");
       ResultSet  rs=(ResultSet)session.getAttribute("rs");
       if(rs != null)
         {
           out.print("数据查询结果");
           out.print("<Table Border>");
           out.print("<TR>");
           out.print("<TH width=100>"+"学号"+"</th>");
           out.print("<TH width=100>"+"姓名"+"</th>");
           out.print("<TH width=50>"+"数学成绩"+"</th>");
           out.print("<TH width=50>"+"英语成绩"+"</th>");
           out.print("<TH width=50>"+"物理成绩"+"</th>");
           out.print("</TR>");
           while(rs.next())
             {
               out.print("<TR>");
               number=rs.getString(1);
               name=rs.getString(2);
               out.print("<TD>"+number+"</TD>");
               out.print("<TD>"+name+"</TD>");
               math=rs.getInt(3);
               out.print("<TD>"+math+"</TD>");
               english=rs.getInt(4);
               out.print("<TD>"+english+"</TD>");
               physics=rs.getInt(5);
               out.print("<TD>"+physics+"</TD>");
               out.print("</TR>") ;
             }
           out.print("</Table>");
         }
       else
         out.println("无数据");
     %>
    </font>
    <br><br>
    <a href = "../LoginSuccess.jsp">返回</A>
</center>
```

```
</BODY>
</HEAD>
</HTML>
```

【说明】

该程序创建一个表单，表单包含一个文本框，客户在此框输入学号。提交该表单后，从 session 中获取结果集 rs，并以表格形式输出该结果集数据。

query_2.jsp 程序源代码如下。

```
<!-- 例程 10-9 query_2.jsp -->

<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import ="java.sql.*" %>
<HTML>
<BODY>
  <%!
     ResultSet chaxun(String number)
     {
       Connection con=null;
       Statement sql=null;
       ResultSet rs=null;
       try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
          }
       catch(ClassNotFoundException e) {}
       try{
            con=DriverManager.getConnection("jdbc:odbc:grade");
            sql=con.createStatement();
            String condition="SELECT * FROM students where number LIKE "+ "'"+number+"'" ;
            rs=sql.executeQuery(condition);
            con.close();
            return  rs ;
          }
         catch(SQLException e)
          { return  rs; }
      }
   %>
  <%
      //获取提交的学号
     String number=request.getParameter("number");
     if(number==null)
       {
           number="";
       }
     number=number+"";
     byte b[]=number.getBytes("ISO-8859-1");
     number=new String(b);
   %>
```

```
  <%
     ResultSet  shu = chaxun(number);
     if(shu == null)
       {
         session.setAttribute("rs", "null");
       }
     else
       {
         session.setAttribute("rs",shu);
       }
    response.sendRedirect("query_1.jsp");
  %>
</BODY>
</HTML>
```

【说明】

该程序中定义了一个 ResultSet chaxun(String number)方法,该方法用于获取学号是 number 的结果集。程序从表单中获取学号 number,然后执行查询,获得结果集数据 shu,如果 shu 不为空,则把属性值("rs",shu)加入 session 中,并重新定向到 query_1.jsp 页面。

10.1.7　删除记录

该模块根据学号删除学生记录,由两个页面组成。在 del_1.jsp 页面中输入学号,提交给 del_2.jsp 页面,del_2.jsp 页面完成学生记录删除。其页面交互关系如图 10-7 所示。

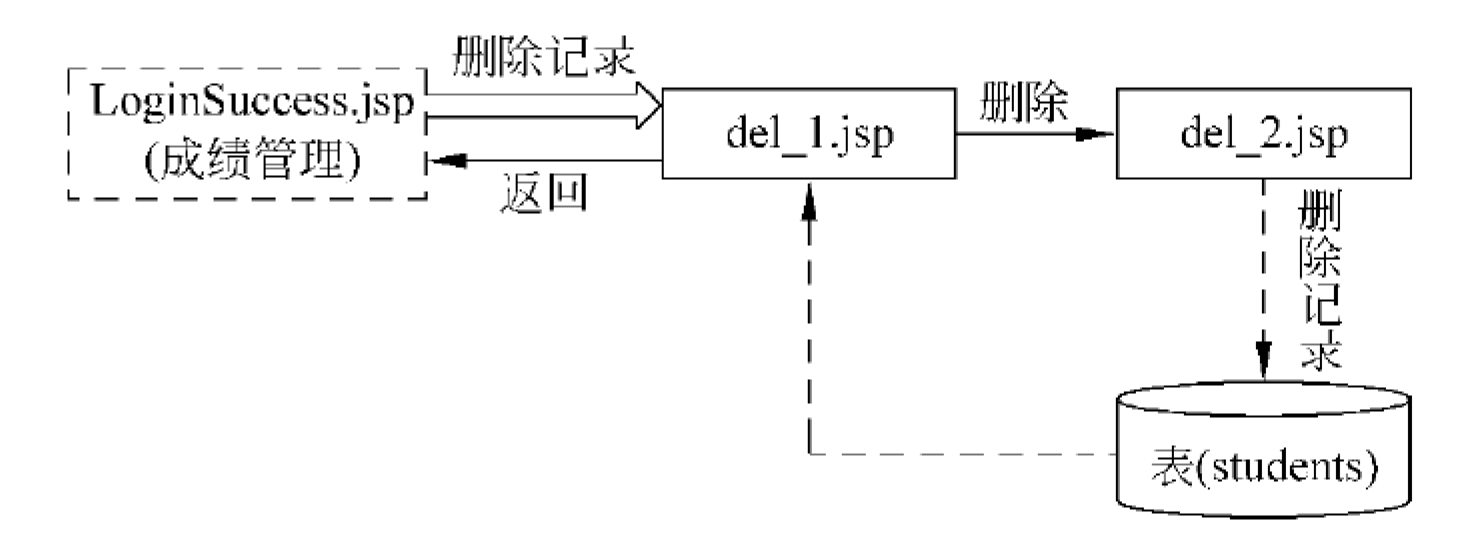


图 10-7　成绩删除模块交互图

del_1.jsp 程序源代码如下。

<! -- 例程 10 - 10 del_1.jsp -- >

```
<%@ page contentType = "text/html;charset = GB2312" %>
<%@ page import = "java.sql.*" %>
<HTML>
<HEAD>
<TITLE>删除记录</TITLE>
<BODY>
<center>
  <font size = 5><b>删除记录</b></font>
  <FONT size = 4>
```

```
        <FORM action = "del_2.jsp" method = post >
            <table >
              <tr >
                  <th  align = left >
                    <font size = 4 > 输入学号:</font > <Input type = "text" name = "number">
                  </th >
              </tr >
              <tr >
                  <th align = center >< br >
                    <Input type = "submit" name = "b" value = "删除">
                  </th >
              </tr >
            </table >
        </FORM >
        <br ></br >
        <%
            String  del = (String)session.getAttribute("del");
            if(del == null) del = "" ;
            out.println(del);
         %>
       <br ></br >
       <a href  =  "../LoginSuccess.jsp">返回</A >
       </FONT >
    </center >
    </BODY >
    </HEAD >
    </HTML >
```

【说明】

该程序创建一个表单,表单中包含一个文本框,客户在此框中输入学号。提交表单后,从 session 中获取删除标志(del)。del 有 3 种取值:“删除成功”、“删除失败”和“没有这个学号”,最后输出删除标志 del。

del_2.jsp 程序源代码如下。

<! -- 例程 10 - 11 del_2.jsp -- >

```
    <%@ page contentType = "text/html;charset = GB2312"  %>
    <%@ page import = "java.sql. * "  %>
    <HTML>
    <BODY>
      <%!
         boolean query(String number)   //查询是否有学号是 number 的学生
         {
           Connection con = null;
           Statement sql = null;
           ResultSet rs = null;
           try{
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
              }
           catch(ClassNotFoundException e) {}
```

```
        try{
            con = DriverManager.getConnection("jdbc:odbc:grade","sa","123456");
            sql = con.createStatement();
            String condition = "SELECT * FROM students where number = " + "'" + number + "'" ;
            rs = sql.executeQuery(condition);
            int num = 0;
            while(rs.next()) num ++ ;
            con.close();
            if(num > 0)
              return  true;
            else
              return  false ;
          }
         catch(SQLException e)
           { return  false; }
       }
  %>
 <%!
     String del(String number) //删除学号是 number 的学生
     {
        if(query(number))
          {
            Connection con = null;
            Statement sql = null;
            ResultSet rs = null;
            try{
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
              }
            catch(ClassNotFoundException event)  {}
            try{
                con = DriverManager.getConnection("jdbc:odbc:grade");
                sql = con.createStatement();
                  //删除操作
                String deleteALL = "DELETE FROM students WHERE number" + " = " + "'" + number + "'";
                sql.executeUpdate(deleteALL);
                con.close();
                return  "删除成功";
              }
             catch(SQLException event)
              {
                 return  "删除失败" ;
              }
            }
          else
            return "没有这个学号";
       }
  %>
 <%
        //获取提交的学号:
        String number = request.getParameter("number");
        number = number.trim();
```

```
            if(number == null)
                {
                  number = "";
                }
            byte b[ ] = number.getBytes("ISO-8859-1");
            number = new String(b);
        %>
        <%
            if(del(number).equals("删除成功"))
                session.setAttribute("del", "删除成功");
            else
                if(del(number).equals("删除失败"))
                   session.setAttribute("del","删除失败");
                else
                   session.setAttribute("del","没有这个学号");
            response.sendRedirect("del_1.jsp");
        %>
    </BODY>
    </HTML>
```

【说明】

该程序定义了两个方法，boolean query(String number)方法用于查询学号是 number 的学生，若查找成功，则返回值为 true，否则返回值为 false；String del(String number)方法则用于删除学号是 number 的学生。根据删除返回的标志向 session 中添加相应的属性值，并重新定向到 del_1.jsp 页面。

10.2 在线考试系统

网络在线考试系统旨在实现考试的无纸化管理，对一些科目的考试可以通过互联网络或局域网进行，方便校方考务的管理，也方便了考生，尤其适合考生分布广，不易集中的远程教育。本系统的考试题由系统从题库中随机抽取，考生做完试题交卷后，系统自动改卷，自动评分。

10.2.1 考试设计原理

1. 系统构成

本系统由 3 个模块组成，分别是产生试卷模块、获取试题模块和改卷模块。其中产生试卷模块用于获取库中的题目、选项、答案；获取试题模块用于产生考题界面；改卷模块交由系统自动改卷，自动评分。

2. 数据库设计

本数据库包含一张表(Exam)，该表用来存放考试题目、每道题的 4 个选项、答案编号。该表结构如表 10-3 所示。

表 10-3　Exam 表结构

字段名	数据类型	字段宽度	字段作用
QID	文本	自动	关键字段
Qus	文本	255	考题
Opt1	文本	50	第 1 选项,对应编号是:1
Opt2	文本	50	第 2 选项,对应编号是:2
Opt3	文本	50	第 3 选项,对应编号是:3
Opt4	文本	50	第 4 选项,对应编号是:4
Ans	文本	1	答案编号,取值:1～4

3. 系统执行流程

本系统执行流程如图 10-8 所示。

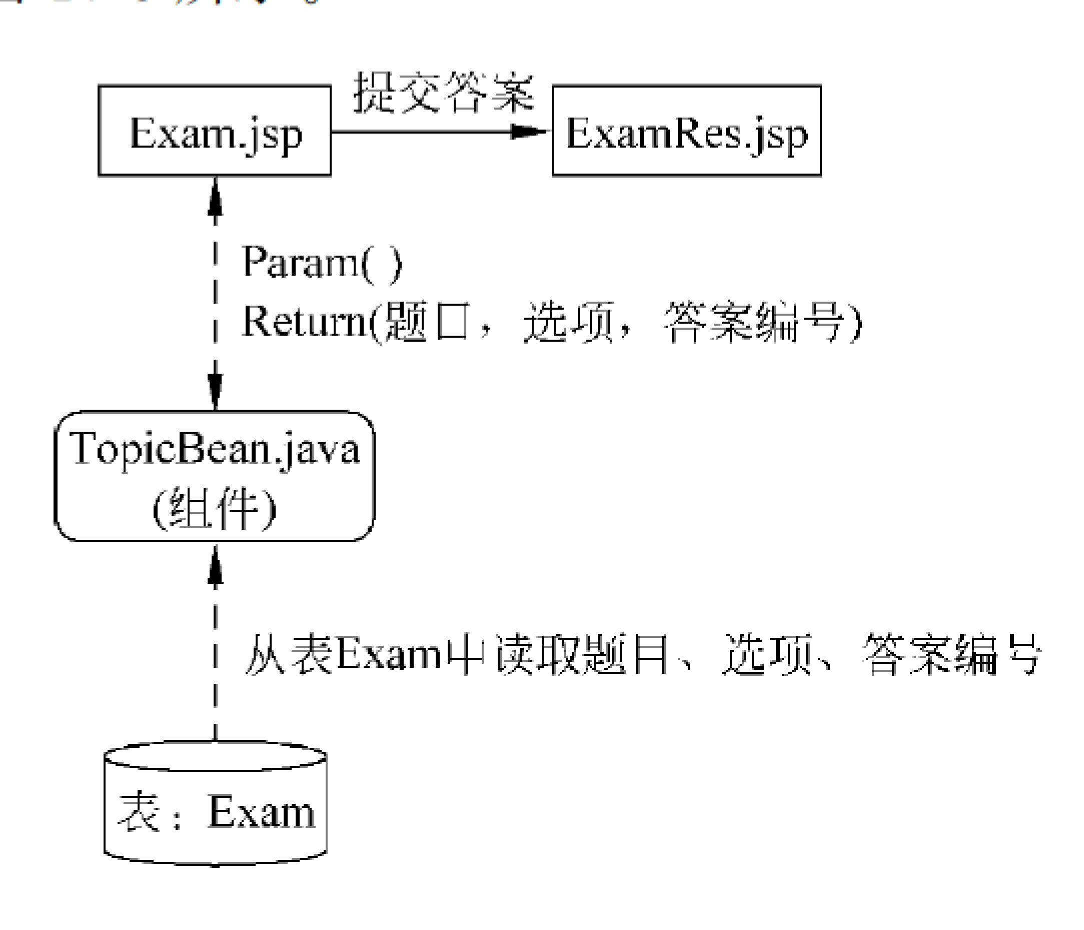


图 10-8　在线考试的执行流程图

10.2.2　产生试卷

本模块由 Exam.jsp 页面实现。它从 Exam 表中获取题目、选项和答案编号,创建试卷,客户在试卷上选择答案。

Exam.jsp 程序源代码如下。

```
<! -- 例程 10 - 12 Exam.jsp -- >

<%@ page contentType = "text/html; charset = GB2312"
   import = "java.sql.* ,   java.util.Random" %>
<%@ page import = "small.dog.* "  %>
<HTML>
  <HEAD>
    <TITLE>产生考题</TITLE>
  </HEAD>
  <BODY>
    <CENTER>
      <FONT SIZE  =  5 COLOR  =  blue>在线考试</FONT>
   </CENTER>
```

```
<HR>
<P></P>
    <jsp:useBean id = "top" class = "small.dog.TopicBean" scope = "page" >
    </jsp:useBean>
    <%!
        ResultSet  rs = null;
    %>
   <FORM action = "./ExamRes.jsp"  method = post  name = FORM1 >
   <%
     rs = top.getRs("SELECT * FROM Exam");
     boolean chcQus[] = new boolean[17];          //设表中有 17 道题目
     int i = 1;
     Random Rnd = new Random();                   //建立产生随机数的 Random 对象
     while(i < 8)                                 //产生 7 道考题
     {
     int chcNum = Rnd.nextInt(16) + 1;            //取得值为 1~17 的随机数
     if(chcQus[chcNum - 1] != true)             //判断表中行号为 chcNum 的记录是否被选取
     {
         chcQus[chcNum - 1] = true;               //将标记设为真,将选取该记录
         rs.absolute(chcNum);                     //游标移到被选取的记录处
         String Ans = rs.getString("Ans");        //取得该记录答案编号
         String Qus = rs.getString("Qus");        //取得该记录的题目
         String Opt = null;
         session.setAttribute("Ans" + i, Ans);    //答案编号存入 session 对象中
         session.setAttribute("Qus" + i, Qus);    //试题存入 session 对象中
     %>
                         <!-- 显示题号 i -->
         <FONT SIZE = 4 COLOR = brown><% = i %>.</FONT>
         <FONT SIZE = 4 COLOR = DarkBlue>
         <!-- 显示题目 Qus --><% = Qus %>
         </FONT><BR>
         <%
         for(int j = 1; j <= 4; j++ )        //利用 for 循环产生选项
         {
             Opt = rs.getString("Opt" + j);
             if (Integer.parseInt(Ans) == j)
                   session.setAttribute("RghOpt" + i, Opt);
          %>
                       <!-- 显示单选按钮,选项值分别是:1,2,3,4 -->
             <INPUT type = "radio" name = rdoQ<% = i%> value = <% = j %>>
             <FONT SIZE = 4>
             <!-- 显示选项内容 Opt -->
             <% = Opt %> </FONT>
             <%
         }
         i = i + 1;        //已产生题数加 1
          %><P></P><%
     }
          }
     %>
    <INPUT type = "submit" value = "提交答案" name = submit1 >
```

```
        </FORM>
</BODY>
</HTML>
```

【说明】

（1）本程序将表单中的数据提交给 ExamRes.jsp 页面。

（2）程序中用类 small.dog.TopicBean 创建一个 Bean，其名称是 top，并声明一个结果集类型的变量 rs，用于获取组件 top 的结果集，该结果集中包含 Exam 表的所有记录。

（3）程序创建了一个布尔型数组 chcQus[]，用来标示表中已被选取的题目。

（4）创建随机数对象 Rnd，通过外循环随机产生 7 道试题。把正确答案编号（Ans）、试题（Qus）、正确答案（Opt），分别以属性名"Ans" + i、"Qus" + i、"RghOpt" + i 保存到 session 对象中。

（5）通过内循环 for(int j = 1; j <= 4; j++)为每道试题产生 4 个单选按钮和选项。

10.2.3　获取试题

本模块是由组件（TopicBean.java）实现的，其功能是从试题库（即 Exam 表）中获取试题和答案（rs）。

TopicBean.java 程序源代码如下。

<! -- 例程 10 - 13 TopicBean.java -- >

```
package small.dog;
import java.sql.*;
public class TopicBean
{
    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;
    public  TopicBean()
    {
        try{
          Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
          con = DriverManager.getConnection("jdbc:odbc:grade");
          stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                 ResultSet.CONCUR_READ_ONLY);
          //建立 Statement 对象，并设置记录指标类型为可前后移动
           }
        catch(Exception ex)
           {
             System.out.println(ex.toString());
           }
     }
     public ResultSet getRs(String sql)
     {
         try
            {
              rs = stmt.executeQuery(sql);
```

```
                return rs;
            }
        catch(SQLException ex)
            {
                System.out.println(ex.toString());
                return   null;
            }
    }
    public void closeBean()        //执行关闭各种对象的操作
    {
        try{
              rs.close();
              stmt.close();
              con.close();
           }
        catch(Exception ex)
          {
               System.out.println(ex.toString());
          }
    }
}
```

【说明】

本程序中定义了 3 个方法，TopicBean()方法用于加载驱动程序、创建连接对象(con)和语句对象(stmt)；ResultSet getRs(String sql)方法用于获取 sql 语句的结果集(rs)；closeBean()方法用于释放各种对象。

10.2.4 批改试卷

本模块提供一个界面 ExamRes.jsp，用于获取客户提交的答案，并与试题库中的正确答案比较来批改试题，最后统计考生的得分。

ExamRes.jsp 程序源代码如下。

```
<!-- 例程 10-14 ExamRes.jsp -->

<%@ page contentType="text/html; charset=GB2312" %>
<HTML>
<HEAD>
<TITLE>阅卷</TITLE>
</HEAD>
<BODY>
<CENTER>
<FONT SIZE = 5 COLOR = blue>在线考试</FONT>
</CENTER>
<HR>
<P></P>
    <%
      int RightAns = 0; //保存答对的题数
      //在下面的 for 循环中，利用 if 判断式进行答案的对比
      for(int i = 1; i<8; i++)
        {
```

```
 %>
<P>第<% = i%>题您
<%
//从 session 对象中取得第 i 道题目的答案(Ans )
String Ans = (String) session.getAttribute("Ans" + i);
//从表单中取得客户对第 i 道题选择的答案(UserAns)
String UserAns = request.getParameter("rdoQ" + i );
if(UserAns == null) //客户没有选择答案
  {
    %>
   <FONT Size = 4 COLOR = RED>未做答</FONT><BR>题目是
   <FONT COLOR = Green><B>

   <! -- 输出题目 -->
   <% = session.getAttribute("Qus" + i) %>
   </B></FONT><BR>答案是<FONT COLOR = Brown><B>

   <! -- 输出答案编号 -->
   [<% = Ans %>]

   <! -- 输出答案所对应的选项 -->
   <% = session.getAttribute("RghOpt" + i) %></B></FONT></P>
   <%
  }
else if(UserAns.equals(Ans))//客户选择了正确答案
      {
      RightAns = RightAns + 1; //答对题数加 1
      %>
      答<FONT Size = 4 COLOR = Blue>对</FONT>了</P>
      <%
        }
     else //客户选择了错误的答案
        {
      %>
      答<FONT Size = 4 COLOR = RED>错</FONT>了<BR>

      题目是<FONT COLOR = Green><B><% = session.getAttribute("Qus" + i) %>
      </B></FONT><BR>
      答案是 <FONT COLOR = Brown><B>

      <! -- 输出答案编号 -->
      [<% = Ans %>]

      <! -- 输出答案所对应的选项 -->
      <% = session.getAttribute("RghOpt" + i) %></B></FONT></P>
      <%
      }
  }
   %>
<H3>七题中您共答对了
<FONT COLOR = RED><! -- 输出答对的题数 --><% = RightAns %>
```

```
    </FONT>题
</BODY>
</HTML>
```

【说明】

本程序声明了一个变量 RightAns，用来保存客户答对试题的数目，通过循环语句遍历试题，批改每道试题。其批改步骤如下：

(1) 从 session 对象中获取正确答案编号(Ans)。

(2) 从表单中获取客户提交的答案编号(UserAns)。

(3) 若客户提交的答案编号为空，即未选择答案(UserAns == null)，则输出：试题，正确答案编号，答案。

(4) 若客户选择的答案与正确答案相同，则输出信息：答对了。

(5) 若客户选择了错误的答案，则输出信息：答错了，试题正确答案编号，答案。

(6) 输出答对题的数目：RightAns。

10.3 问卷调查

问卷调查是指以试卷的方式调查客户意见，并对调查结果作出统计。

10.3.1 问卷设计原理

1. 系统构成

本系统由 4 个模块构成，分别是问卷界面模块、数据库连接模块、保存问卷记录模块和查看问卷结果模块。各模块的作用如下所述。

- 问卷界面模块(usFrm.jsp)：产生问卷调查界面，客户在此界面选择选项。
- 数据库连接模块(DBCon.java 组件)：建立与数据库的连接。
- 保存问卷记录模块(QusBean.java 组件)：保存问卷题目、选项、问卷记录。
- 查看问卷结果模块(QusRes.jsp)：从库中获取调查结果，并加以统计显示。

2. 数据库设计

本系统包含一张问卷表(Qus)，用于保存问卷数据。该表的结构如表 10-4 所示。

表 10-4 问卷表结构

字段名	数据类型	字段宽度	字段作用
ID	文本	自动	关键字段
Qus1	数字	自动	第一题选项值
Qus2	数字	自动	第二题选项值
Qus3	数字	自动	第三题选项值

在该问卷系统中包括 3 道题目，每一道题有 5 个选项，当客户选择第 1 选项时，其选项值就是 1，以此类推，当客户选择第 i 个选项时，其选项值就是 i。i 的取值是 1～5。

10.3.2　创建问卷界面

该模块产生一个问卷界面，客户在此界面选择自己对问题的答案。该模块用 QusFrm.jsp 页面来实现，QusFrm.jsp 页面从 Qus 组件（由类 QusBean.class 创建）中获取问题和选项，创建问卷界面。然后把客户选择的答案提交给 Qus 组件。由 Qus 组件把客户选择的答案保存到数据表 Qus 中。本页面与其他页面（组件）的交互关系如图 10-9 所示。

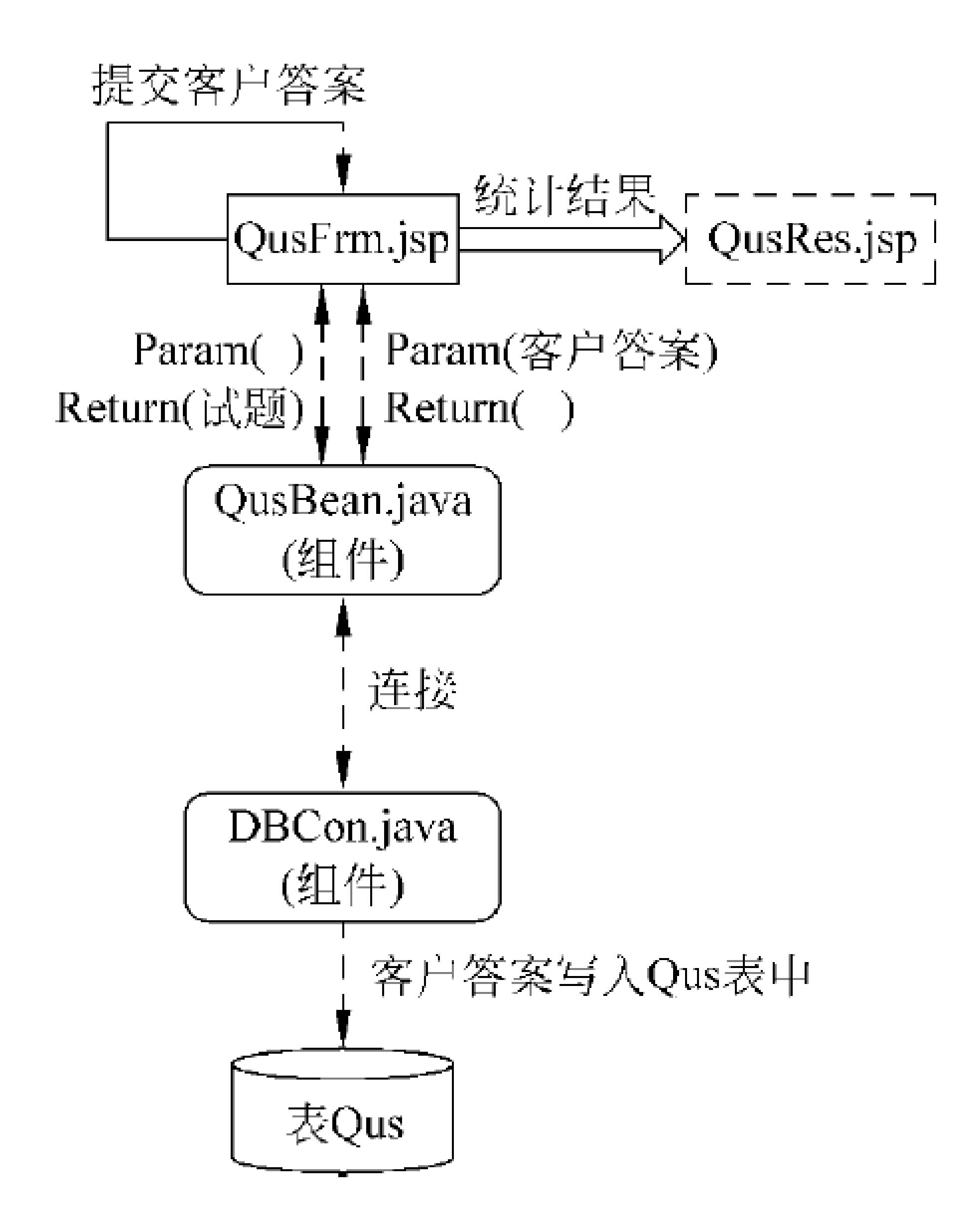


图 10-9　创建问卷界面交互图

QusFrm.jsp 页面程序源代码如下。

```
<! -- 例程 10 - 15 QusFrm.jsp -- >

<% @ page contentType = "text/html; charset = GB2312" %>
<HTML>
<HEAD><TITLE>问卷调查</TITLE></HEAD>
<BODY>
<CENTER>
<FONT SIZE = 5 COLOR = blue>问卷调查</FONT>
</CENTER>
<HR>
<P>
  <jsp:useBean id = "qus" scope = "session" class = "small.dog.QusBean"/>
  <%
    if(request.getParameter("submit1") == null)  //若没有提交表单则创建问卷界面
      {
      %>
      <FONT size = 4 >请告诉我们您对<BR></FONT>  <B>
      <FONT size = 5 Color = midnightblue>

      <% -- 显示 title 属性值 -- %>
      <jsp:getProperty name = "qus" property = "title"/>
```

```
        </FONT>
        <FONT size = 4 Color = green>

        <% -- 显示 subTitle 属性值 -- %>
        <jsp:getProperty name = "qus" property = "subTitle"/>
        </FONT> </B><BR>
        <FONT size = 4>—书的意见</FONT></P><BR>
        <FORM action = "QusFrm.jsp" method = Post>
        <%
            //第一层 for 循环用于输出问卷题目
        for(int i = 0; i < 3; i++)
        {
         %>
        <FONT size = 4 color = Gray><B><% = qus.getQus(i) %>
        </B></FONT><BR><BR>
        <%
          //第二层 for 循环用于输出问卷各题目的选项
        for(int j = 0; j < 5; j++)
        {
            %>
            <INPUT type = "radio" name = radio<% = i + 1 %>
                value = <% = j + 1 %>>
            <FONT size = 4 color = Brown><% = qus.getOpt(j) %>
            </FONT>
        <%
        }
         %><BR><BR><BR>
        <%
        }
         %>
        <BR>
        <INPUT  type = "Submit"   name = submit1   value = "提交客户答案" >   
        <A href = QusRes.jsp>统计结果</A>
        </FORM>
        <%
        }
      else       //若提交了表单数据,则将数据增加到数据表中
       {  %>
        <jsp:useBean id = "con" scope = "session"   class = "small.dog.DBCon"/>
        <%
            qus.InsertAns(con.getConnection(),
            request.getParameter("radio1"),
            request.getParameter("radio2"),
            request.getParameter("radio3"));
         %>
        <P><FONT SIZE = 5 COLOR = red>非常感谢您的意见</FONT></P>
        <A href = QusRes.jsp>统计结果</A>
        <%
       }
 %>
</BODY>
</HTML>
```

【说明】 本程序中用 small.dog.QusBean 类创建组件 qus,若提交控件(submit1)的值为空,即还未创建问卷界面,则输出组件 qus 的标题和副标题的属性值,输出问卷题目和选项,以此来创建问卷界面,否则,就将问卷数据加入数据库表 Qus 中。

10.3.3 保存问卷答案

本模块由两个组件来完成,功能是把客户的答案保存到数据库表中。其中,DBCon.java 组件用于实现数据库连接,而 QusBean.java 组件有两个方面的作用:一方面保存问卷题目和每个题目的选项,用于创建问卷界面;另一方面是将用户选择的问卷答案加入到数据库表 Qus 中。

本模块程序源代码如下。

<! -- 例程 10 - 16 DBCon.java -- >

```
package small.dog;                              //定义 Bean 所属的包
import java.io.*;
import javax.servlet.http.*;
import java.sql.*;

//定义 DBCon 类实现 HttpSessionBindingListener 接口
public class DBCon implements HttpSessionBindingListener
{
  private Connection con = null;
  public DBCon()
  {
    BulidConnection();                          //建立库连接
  }
  private void BulidConnection()                //建立库连接的方法
  {
    try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con = DriverManager.getConnection("jdbc:odbc:grade");
    }
    catch(Exception ex)
    {
        System.out.println(ex.toString());
    }
  }
  public Connection getConnection()             //获取连接对象的方法
  {
      //若 con 为 null 时,重新建立库连接
    if(con == null)
        BulidConnection();
    return this.con;
  }
  public void close()
  {
```

```
    try{
        con.close();
        con =   null;
         }
    catch(SQLException sex)
         {
        System.out.println(sex.toString());
         }
   }

   //当有对象加入 session 中时,将自动执行此方法
   public void valueBound(HttpSessionBindingEvent event){}

   //当有对象从 session 中删除时,将自动执行此方法
   public void valueUnbound(HttpSessionBindingEvent event)
   {
     if(con != null)
       close();
   }
}
```

<! -- 例程 10 - 17 QusBean.java -- >

```
package small.dog;
import java.sql.*;
public class QusBean
{
    private String Title = "JSP 编程技术";                          //书籍名称
    private String SubTitle = "上机练习与实际应用(附光盘)";   //书籍副标题
    private int QusNum = 3;    //题目数
    private String Qus[] = {  "您满意本书的内容吗?",
                             "您满意本书的版面编辑吗?",
                             "您满意该书的封面设计吗?"  };      //题目
    private int OptNum = 5;                                         //选项个数
    private String Opt[] = {"很满意","满意","尚可","不满意","很不满意"};
    private String Color[] = {"Yellow","DeepPink","DarkCyan","DeepSkyBlue",
            "Orange","LightSlateGray"};    //定义产生图表的颜色

    //以下的方法获取组件的属性
    public String getQus(int i)
    { return Qus[i]; }
    public String getOpt(int i)
    { return Opt[i]; }
    public String getColor(int i)
    { return Color[i]; }
    public String getTitle()
    { return this.Title; }
    public String getSubTitle()
    { return this.SubTitle; }

    //与数据库连接有关的属性
```

```
    private Connection con = null;
    private Statement stmt = null;
    public QusBean()
    {
    }

    //将问卷的答案加入数据库表中，返回值为加入的记录笔数
    public int InsertAns(Connection con, String rad1,String rad2, String rad3)
    {
    int affect = 0;
    String strSQL = "INSERT INTO Qus(Qus1, Qus2, Qus3) " +
           "VALUES(" + rad1 + "," +   rad2 + "," + rad3 + ")";
    try{
        this.con = con;
        stmt = this.con.createStatement(
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_READ_ONLY);
        affect = stmt.executeUpdate(strSQL);
       }
    catch(SQLException sex)
       {
        System.out.println(sex.toString());
       }
    return affect;
  }
  public int RecNum(Connection con, String filter)     //取得记录总数
  {
      String strSQL = "SELECT ID FROM Qus";
      int num = 0;
      if(!filter.equals(""))                          //判断 filter 字符串是否为空白字符串
      {
        strSQL = strSQL + " WHERE " + filter;
      }
      try{
        Statement lstmt = con.createStatement(
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_READ_ONLY);
        ResultSet rs = lstmt.executeQuery(strSQL);   //执行查询
        rs.last();                                    //移至最后一条记录
        num =  rs.getRow();                           //取得记录总数
        rs.close();
        lstmt.close();
      }
    catch(SQLException sex)
      {
        System.out.println(sex.toString());
      }
    return num;
    }
}
```

专家点拨：两个类文件(DBCon.class、QusBean.class)部署在 small.dog 包中。

10.3.4 查看问卷结果

此模块由一个 QusRes.jsp 页面实现，其作用是从数据库中获取客户的答卷数据并加以统计，用条状图方式输出问卷结果。

QusRes.jsp 程序源代码如下。

```
<! -- 例程 10 - 18 QusRes.jsp -- >

<%@ page contentType = "text/html; charset = GB2312" %>
<HTML>
<HEAD><TITLE>问卷调查</TITLE></HEAD>
<BODY>
<CENTER>
<FONT SIZE  =  5 COLOR  =  blue>问卷调查</FONT>
</CENTER>
<HR>
  <jsp:useBean id = "qus" scope = "session" class = "small.dog.QusBean"/>
  <jsp:useBean id = "con" scope = "session" class = "small.dog.DBCon"/>
  <FONT size = 5 Color = midnightblue>
     <jsp:getProperty name = "qus" property = "title"/>
  </FONT>
  <FONT size = 4 Color = green>
     <jsp:getProperty name = "qus" property = "subTitle"/>
  </FONT>
  <FONT size = 4> 问卷统计结果... </FONT></B>
  <%
     int TotalRec  =  qus.RecNum(con.getConnection(), "");
     //取得 Qus 表中的总记录笔数, 此值相当于总问卷数
     int Count  =  0, PRad  =  0;

    //第一层 for 循环用于输出问卷题目
   for(int i  =  0; i < 3; i++ )
   {
     %>
    <BR></BR>
    <FONT size = 4 color = Gray><B><%  =  qus.getQus(i) %></B></FONT>
    <BR></BR>
    <%
    //第二层 for 循环用于输出问卷各选项统计
    for(int j  =  0; j < 5; j++ )  {
     %>
    <TABLE><TR width = 290>
    <TD width = 120><FONT size = 4 color = Maroon><B>
    <! -- 从 aryOpt 阵列取得选项值 -->
    <%  =  qus.getOpt(j) %></B></FONT></TD>
    <%
    Count  =  qus.RecNum(con.getConnection(),"Qus"  +  (i + 1)  +  "  =  "  +  (j + 1));
    //取得记录集中第 i 题、第 j 选项被选中的次数
```

```
        PRad = Count * 100 / TotalRec;     //计算第 i 题、第 j 选项被选取的百分比
        %>
        <!--
        以百分比值作为储存格的宽度，该格的颜色从 aryColor 阵列中取得
        -->
        <TD width=<%= PRad %> bgcolor=<%= qus.getColor(j) %>></TD>
        <TD><FONT size=4 color=Red><B>(<%= PRad %>%)</B></FONT>
        </TD>
        </TR></TABLE>
        <%
        }
      }
    %>
  </BODY>
  </HTML>
```

【说明】

本程序用 small.dog.DBCon 类创建连接对象 con，用 small.dog.QusBean 类创建问卷对象 qus。

本章小结

本章用具体实例介绍了 3 个应用系统的编程思路，它们综合运用了 JSP 技术和方法。通过本章的学习，使读者加深对相关的概念、方法和技术的理解和灵活运用。

相关课程教材推荐

ISBN	书　名	作　者
9787302184287	Java 课程设计(第二版)	耿祥义
9787302131755	Java 2 实用教程(第三版)	耿祥义
9787302135517	Java 2 实用教程(第三版)实验指导与习题解答	耿祥义
9787302184232	信息技术基础(IT Fundamentals)双语教程	江　红
9787302177852	计算机操作系统	郁红英
9787302178934	计算机操作系统实验指导	郁红英
9787302179498	计算机英语实用教程(第二版)	张强华
9787302180128	多媒体技术与应用教程	杨　青
9787302177081	计算机硬件技术基础(第二版)	曹岳辉
9787302176398	计算机硬件技术基础(第二版)实验与实践指导	曹岳辉
9787302143673	数据库技术与应用——SQL Server	刘卫国
9787302164654	图形图像处理应用教程(第二版)	张思民
9787302174622	嵌入式系统设计与应用	张思民
9787302148371	ASP.NET Web 程序设计	蒋　培
9787302180784	C++程序设计实用教程	李　青
9787302172574	计算机网络管理技术	王　群
9787302177784	计算机网络安全技术	王　群
9787302176404	单片机实践应用与技术	马长林

以上教材样书可以免费赠送给授课教师，如果需要，请发电子邮件与我们联系。

教学资源支持

敬爱的教师：

感谢您一直以来对清华版计算机教材的支持和爱护。为了配合本课程的教学需要，本教材配有配套的电子教案(素材)，有需求的教师可以与我们联系，我们将向使用本教材进行教学的教师免费赠送电子教案(素材)，希望有助于教学活动的开展。

相关信息请拨打电话 010-62776969 或发送电子邮件至 weijj@tup.tsinghua.edu.cn 咨询，也可以到清华大学出版社主页(http://www.tup.com.cn 或 http://www.tup.tsinghua.edu.cn)上查询和下载。

如果您在使用本教材的过程中遇到了什么问题，或者有相关教材出版计划，也请您发邮件或来信告诉我们，以便我们更好为您服务。

地址：北京市海淀区双清路学研大厦 A 座 708　　计算机与信息分社魏江江　收
邮编：100084　　电子邮件：weijj@tup.tsinghua.edu.cn
电话：010-62770175-4604　　邮购电话：010-62786544